IPODLAS
A Framework for Coupling Temporal Simulation Systems, Virtual Reality, and Geographic Information Systems

Dissertation

zur

Erlangung der naturwissenschaftlichen Doktorwürde
(Dr. sc. nat.)
vorgelegt der
Mathematisch-naturwissenschaftlichen Fakultät
der
Universität Zürich
von

Dani Isenegger
von
Hohenrain LU

Promotionskomitee
Prof. Dr. Robert Weibel (Vorsitz)
Prof. Dr. Walter Schaufelberger (ETHZ)
Dr. Britta Allgöwer (Leitung der Dissertation)

Zürich 2006

Summary

Environmental processes act in space and time and often vary over several spatial and temporal scales. Current software applications dealing with these processes emphasize properties specific to their particular application domain and tend to neglect other concerns. *Temporal simulation systems* (TSS) provide simulation models representing dynamic dependencies of complex processes but typically treat their spatial dimension only poorly. *Virtual reality* (VR) systems, on the other hand, offer photo-realistic 3-D visualization and high-level user interaction but disregard the link between the underlying data and the visualization. *Geographic information systems* (GIS) possess powerful functionality for spatial analysis, data integration, and storage, but favor a static view, generally lacking the representation of dynamics.

This thesis presents *IPODLAS*, standing for **i**nteractive, **p**rocess **o**riented, **d**ynamic **l**andscape **a**nalysis and **s**imulation. To represent spatiotemporal, cross-scale processes IPODLAS applies functionality of the three domains TSS, VR, and GIS comprehensively by exploiting their complementary strenghts. Addressing methodological and experimental aspects the *research questions* of the thesis focus on the design activity of developing the IPODLAS system. Firstly, the development methodology to evolve a system such as IPODLAS is addressed. Secondly, it is studied whether standard GIS can provide the required spatial functionality. Finally, the design concepts and the software architecture of the IPODLAS system are specified.

The main contributions of this research are the development methodology called IPODLAS approach, and the design of the software architecture for the IPODLAS system. The *IPODLAS approach* specifies a framework consisting of three *case studies*. Each provide data and simulation models on different scales, thus supporting a scale-sensitive representation of cross-scale processes. *Use cases* defined within the case studies capture the user requirements and are applied to derive the *functionality listings* specifying the functionality required to satisfy the user requirements. The IPODLAS approach constitutes a methodology for software development projects where integration of functionality from different domains and the consideration of scale is crucial. The functionality listings allow the specification of the required functions contributed by applications of the different domains, i.e. in particular the required GIS functionality.

The software achitecture of the *IPODLAS system* has been developed applying the IPODLAS approach. The IPODLAS system embeds applications of the three domains as subsystems in a common software environment. Applying the *blackboard architecture* a central subsystem is designated to control the communication and synchronization of the IPODLAS system. Providing *mediating functionality* the central subsystem receives *messages* requesting services and dispatches them to the appropriate subsystem(s). The messages exchanged are encoded in XML and form the *communication protocol* of the IPODLAS system. The communication between the subsystems is realized by deploying communication interfaces on each subsystem establishing socket connections between them. The implementation of the software architecture in different evolving prototypes was used as proof of concept and for performance measurements.

The thesis contributes a operational approach of dealing with interdisciplinary problems by combining functionality of the involved domains. The shift of focus from stand-alone applications to a network-centric approach and the growing consideration of interoperability standards form the technical background of this research. Regarding the thesis in this context the IPODLAS system constitutes a general *communication model* for distributed systems in a heterogeneous software environment.

Zusammenfassung

Natürliche Prozesse weisen räumliche und zeitliche Aspekte auf und variieren oft über mehrere räumliche und zeitliche Skalen. Softwareapplikationen, die für die Behandlung solcher Prozesse eingesetzt werden, können gut mit Problemen umgehen, die in ihr spezifisches Anwendungsgebiet fallen, neigen aber dazu, andere Aspekte zu vernachlässigen. Simulationsapplikationen, die zeitorientierte Probleme modellieren (*Temporal simulation systems*, TSS), haben ihre Stärken in der Simulation von dynamischen Beziehungen, vernachlässigen typischerweise aber deren räumliche Dimension. Visualisierungsapplikationen (*Virtual reality*, VR) bieten natürlich wirkende 3D-Visualisierungen und ausgereifte Interaktionsmöglichkeiten, die Visualisierung erfolgt aber getrennt von den unterliegenden Daten. Geographische Informationssysteme (*Geographical informations systems*, GIS) haben ihre Stärken in der Analyse räumlicher Daten sowie deren Integration und Speicherung, haben aber einen statischen Blickwinkel und vernachlässigen die Repräsentation von Veränderungen.

Diese Dissertation präsentiert *IPODLAS*, was für "**i**nteractive, **p**rocess **o**riented, **d**ynamic **l**andscape **a**nalysis and **s**imulation" steht. Für die Repräsentation von raumzeitlichen, skalenübergreifenden Prozessen nützt IPODLAS die komplementären Stärken der Funktionalität aus den Gebieten TSS, VR und GIS. Die *Forschungsfragen* der vorliegenden Arbeit behandeln methodische und praktische Gesichtspunkte des Designs von IPODLAS. Zum Ersten wird auf die Methodologie eingegangen, die für die Entwicklung von IPODLAS verwendet wurde. Zweitens wird untersucht, ob Standard-GIS die dazu benötigte räumliche Funktionalität anbieten können. Die Designkonzepte und die Softwarearchitektur von IPODLAS sind Thema der dritten Forschungsfrage.

Die wichtigsten Ergebnisse dieser Dissertation sind die Entwicklungsmethodologie IPODLAS Approach und das Design der Softwarearchitektur des IPODLAS Systems. Der *IPODLAS Approach* beinhaltet ein Framework bestehend aus drei *Case studies*. Jede bietet Simulationsdaten und –modelle auf verschiedenen Massstabsebenen, wodurch die Repräsentation von skalenübergreifenden Prozessen unterstützt wird. Innerhalb der Case studies wurden *Use cases* entworfen, welche Benutzeranforderungen spezifizieren. Ausgehend von den Use cases wurden *Functionality listing*s generiert, welche die von den Benutzern verlangten Funktionen auflisten. Der IPODLAS Approach bietet eine Methodologie für Softwareprojekte an, welche Funktionalitäten von verschiedenen Gebieten und Massstabsebenen integrieren. Die Functionality listings spezifizieren die benötigten Funktion, welche die involvierten Anwendungsgebiete beisteuern müssen, dh. inbesondere die erforderlichen GIS-Funktionen.

Der IPODLAS Approach wurde verwendet, um die Softwarearchitektur des *IPODLAS Systems* zu entwickeln. Das IPODLAS System bettet Applikationen der verschiedenen Anwendungsgebiete als Subsysteme in eine Softwareumgebung ein. Der *Blackboard-Architektur* entsprechend regelt ein zentrales Subsystem die Kommunikation und Synchronisation des IPODLAS Systems. Das zentrale Subsystem empfängt (*mediates*) und vermittelt Messages zum zuständigen Subsystem. Die XML-codierten Messages bilden das *Kommunikationsprotokoll* des IPODLAS Systems. Die Kommunikation wird durch Schnittstellen realisiert, welche auf die jeweiligen Subsysteme verteilt sind und diese über Sockets verbinden. Die Softwarearchitektur wurde in verschiedenen Prototypen implementiert und für Laufzeitmessungen verwendet.

Die Dissertation zeigt einen operativen Ansatz bei welchem interdisziplinäre Probleme durch Kombiniation von Funktionalität der involvierten Anwendungsgebiete behandelt werden. Die Verschiebung des Fokus von isolierten hin zu vernetzten Applikationen und die wachsende Bedeutung von Interoperabilitäts-Standards bildet den technischen Hintergund dieser Arbeit. In diesen Kontext gestellt bildet IPODLAS ein allgemeines *Kommunikationsmodell* für verteilte Systeme in heterogenen Softwareumgebungen.

Acknowledgments

Prof. Dr. Robert Weibel for his extensive interest, advice, and support to guide and review my thesis which had a major impact on my research.

Dr. Britta Allgöwer for her tremendous amount of inspiring ideas, for many fruitful discussions, and for giving me the opportunity to conduct my PhD within the IPODLAS project.

Prof. Dr. Walter Schaufelberger and Dr. Lars Bernard for reviewing this dissertation.

The IPODLAS team for the sound collaboration and stimulating discussions which caused many innovative inputs for my work. In particular I would like to thank Dr. Andreas Fischlin for his critical analyses and Dr. Urs Frei for his calming influence. Especially I am thankful for the solid collaboration with my fellow IPODLAS PhD students Bronwyn Price and Yi Wu, which helped me through the ups and downs of the IPODLAS project.

Dr. Patrick Laube for critical, but inventive disscussions and suggestions while being in Zurich and in particular for his reviews and proof-reading of my work while equipping sheep with GPS down under.

Dr. Bronwyn Price for proof reading of papers and the dissertation and for providing many helpful suggestions for improving the manuscripts.

Moritz Neun for many helpful arguments and discussions concerning technical issues of my work and for his reviewing of the conceptual part of the PhD.

My (ex-)office colleagues Ronald Schmidt, Joël Fisler, Stefan Steiniger, Nikos Koutsias, Patrick Laube, and Stefan Hofstetter for providing countless hints, tips, and tricks to overcome the various challenges which have to be faced when doing a dissertation.

Friends and colleagues of the GIS and the Geography department for the friendly and convenient atmosphere for working and other hobbies.

My family for patiently giving support in every situation.

The financial support of the Swiss National Science Foundation within the National Research Program 48 'Landscapes and Habitats of the Alps' (Contract no. 4048-064432) and the support of the University of Zurich are gratefully acknowledged.

Contents

Glossary

API	Application Programming Interface
DB	Database
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DBMS	Database management system
DCOM	Distributed Component Object Model
EJB	Entreprise Java Beans
GIS	Geographic Information System
GML	Geography Markup Language
GRASS	Geographic Resources Analysis Support System
HLA	High Level Architecture for Modeling and Simulation
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IC	IpodlasKernel
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IK	IpodlasKernel
IM	ipodlasMessage
IPC	Inter-Process Communication
IPODLAS	Interactive, process oriented, dynamic landscape analysis, and simulation
ISO	International Organization for Standardization
ISO/TC 211	Technical Committee 211 of the International Organization for Standardization
IVR	Immersive Virtual Reality Systems
J2EE	Java 2 Platform, Enterprise Edition
LBM	Larch Bud Moth
MMI	Man-made infrastructure
OGC	Open Geospatial Consortium
OMG	Object Management Group
ORB	Object Request Broker
PDF	Portable Document Format
RAMSES	Research Aids for Modeling and Simulation of Environmental Systems
RASS	RAMSES Simulation Server for Unix Workstations
RDBMS	Relational database management system
RPC	Remote Procedure Call
SciVis	Scientific visualization
SDSS	Spatial Decision Support Systems
SNP	Swiss National Park
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TSS	Temporal Simulation Systems
UDDI	Universal Description, Discovery, and Integration
UI	User interface
UML	Unified Modeling Language

UP	Unified Process
VR	Virtual Reality
VTP	Virtual Terrain Project
W3C	World Wide Web Consortium
WLF	Wildland fire
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language

Part I

The thesis is divided into three parts to separate theoretical considerations required for the development of IPODLAS compiled in Part I from the experimental Part II, where the theoretical concepts and principles are employed to develop and implement IPODLAS. Part III consists of the discussion and the conclusions.

1 Introduction

Processes affecting and forming the landscape are seldom confined to one spatial or temporal scale. Erosion, for instance, takes place as countless microscopic individual events in very small spots, but it is one of the main forces affecting the formation of the whole landscape, particularly in mountainous regions. Avalanches may originate from small release areas, but impact on much larger areas within their avalanche paths. On the temporal scale, avalanches take place in a few seconds, but they affect the landscape and the plants in the avalanche path over several years. The distribution of high and low air pressure over Europe together with topographic structures of the surface of the Earth may cause the formation of Föhn wind systems (a katabatic wind) (Kuhn, 1989) in the European Alps, which increases the temperature in valleys located on the leeward side of the mountains. Landscapes are constantly changing, not only in time but also in space. The relevant processes can be fast and easy to confine in space and time, or slow and hardly noticeable. Some processes can have impacts over the whole Alpine Arc or only affect a single point in a valley. Some processes are discrete, some are continuous; the processes can be man-made or natural (Allgöwer et al., 2001).

Scale

In general modeling environmental processes is a complex task due to the interplay of many variables and changes in space and time. Moreover, natural processes are often interlinked at varying temporal and spatial scales (Peuquet, 2000). Aside from being aware of the classical pitfalls of spatial data — autocorrelation, Modifiable Areal Unit Problem (MAUP), nonuniformity of space and edge effects (O'Sullivan and Unwin, 2002) — considering scale is particularly crucial when working with natural processes. The scale of the observation may affect the representation we use and is likely to have effects on analysis, modeling, and visualization of the phenomena of interest. The definition of scale varies greatly between research communities. In Landscape Ecology spatial scale may involve a measure of the size of patches of particular habitats within the landscape. For a cartographer, metric scale is the ratio between distance on a map and distance on the ground (Quattrochi and Goodchild, 1997).

In the following the term *spatial scale* is used to reference both the magnitude of the area under consideration (spatial extent) and also the degree of detail (spatial resolution or spatial grain) (Quattrochi and Goodchild, 1997). *Temporal scale* is defined in an analogous manner using temporal extent and temporal resolution. Scale is an important factor in Environmental Science, since it is often a parameter influencing processes, which form natural phenomena. For instance, the ratio of extent to resolution of a survey determines the volume of collected data. There are processes that depend only on properties of one point in space or time, and then their variability is only influenced by properties of this

current point and by independent variables. Other processes are in addition influenced by the properties of the local neighborhood in the spatial dimension and by past events in the temporal dimension. For example, in geographic models distance-decay functions can model the impact of neighboring properties on the current observation point with a linear geographic measure (Quattrochi and Goodchild, 1997). In Ecology the hierarchy theory states that spatial and temporal scales tend to covary: processes which operate on large spatial scales also often affect systems over a long temporal scale, partially because space and time are linked trough transport mechanism (Johnson, 1996; Quattrochi and Goodchild, 1997).

A pivotal motivation of the project in which this thesis is embedded is the study of environmental processes extending across the scales and thus, across data, time, and, models. The *Scale-Cube* (Allgöwer et al., 2001) in **Figure 1-1** offers a framework for classifying the modeling of spatiotemporal processes. Extending the idea of the spatiotemporal 'Stommel diagram' (Stommel, 1963) the Scale-Cube is crossed by axes which denote the spatial, the temporal and the model scale. The spatial scale ranges from small to large. The notion of spatial scale is not conceived as a cartographic concept, but relates to spatial extent and resolution. The temporal scale ranges from fast processes with high temporal change frequencies to processes spanning over a long time period with a lower temporal resolution. This classification scheme assumes that when increasing the (spatial or the temporal) scale of a process, both the extent and the resolution are increased. The third axis spans from theoretical over semi-empirical to empirical models. Processes using high spatial resolution data with a short monitoring interval, described by a detailed theoretical model, are located near the origin of the Scale-Cube. Large-scale phenomena with a coarser spatiotemporal resolution, which are often represented by models with a strong empirical background, are positioned at the opposite corner of the cube. Extents of environmental processes can reach from centimeters to kilometers, and seconds to years, and are described depending on the knowledge and goals of the modelers by theoretical, semi-empirical, and empirical models (Steyaert, 1993). Thus, enviromental processes may be located at any place in the Scale-Cube.

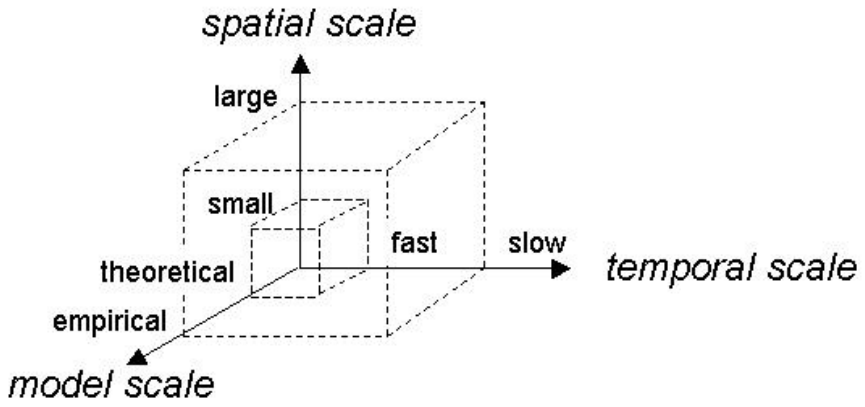

Figure 1-1 The Scale-Cube, crossed by the axes spatial, temporal and model scale, after Allgöwer et al. (2001). The Scale-Cube can be used to classify spatiotemporal processes according to the spatial, temporal, and model scale applied in their models.

Case studies

In the project in which this thesis is embedded, three case studies are investigated which offer realistic data and simulation models. The *Larch Bud Moth* (LBM) case study deals with insect population dynamics, the *wildland fire modeling* (WLF) case study is an example of an abiotic process, and the *wildland fire visualization* (WFV) represents a case study where the focus is on 3-D photorealistic visualization of spatiotemporal, cross-scale processes. The distribution and dynamics of the LBM across the Alpine Arc is an effect of a process taking place on different spatial and temporal scales. To describe the LBM dynamics in space and time from the valley scale to the level of the whole Alpine Arc several models with different modeling approaches are applied, the modeled temporal extent spans from years to centuries (Fischlin, 1991). In the WLF case study different Wildland fire models describe surface and crown fire, which usually occurs at a larger spatial scale than a surface fire, and are located at different spots within the Scale-Cube. The temporal scale for WLF spans from minutes to days (Finney, 1998). The WFV case study spans from wildland fire visualization at the valley scale to flame visualization at the local scale; it also covers the visualization of transitions between the two extremes. The case studies and the applied simulation models are described in greater detail in section 5.1.

A Vision …

The understanding of spatiotemporal, cross-scale processes and their interrelations is central to the understanding of the complex behavior of real world systems (Pang and Shi, 2002). The knowledge of spatiotemporal, cross-scale environmental phenomena may be enhanced when tools for conceptual representation and analysis, modeling, and visualization also take into account varying scales in space (Johnson, 1996) and time and different models. But what kind of tool can be envisaged to meet these requirements?

A vision of an *ideal framework* to investigate, for instance, the LBM dynamics over the Alpine Arc consists of an interactive tool combining realistic visualization with spatiotemporal analysis and simulation capabilities. The application allows navigation in 3D over the landscape and zooming in and out to relevant regions allowing visual exploration of the areas of interest. Additional data requested by the user, for example the larch distribution over the Alpine Arc, can be selected and displayed. An overview of available information concerning the phenomena of interest is presented to the user. This can be textual or graphical data, details of simulation models or multimedia information. The framework supports user friendly modeling in several ways. It presents the applicable simulation models, provides usage information and metadata, and allows exploration of the model properties and parameters. Advanced users are supported in manipulation of model variables and parameters. Models operating on different spatial or temporal scales or models simulating different aspects probably stemming from different fields of application relevant to LBM dynamics can be coupled to obtain a more holistic, cross-scale representation. The data and simulations can be explored across their temporal extent by visualizing changes of spatial patterns chronologically in a movie-like animation. Thus, the exploration of the temporal dimension is enhanced by the ability to move back and forth in time in the animation examining the representation of the process under consideration in its natural environment. Data, simulation results, and also specific navigation paths exploring the landscape can be stored as text files, tabular or multimedia data. Thus, scenario development is supported through comparison of results of model runs using different input variables or parameters or different simulation models.

… and the reality

But now back to reality. Which existing tools can be used to build this envisioned framework? A substantial spatial aspect can be provided by *Geographic Information System* (GIS) and *Virtual Reality* (VR) systems. GIS provides powerful functionality for collection, spatial analysis, data integration, storage and displaying spatial data from different sources (Fedra, 1993; Pang and Shi, 2002). "GIS focus on representations of location, the spatial distribution of phenomena and their relationships to one another in space" (Brimicombe, 2003, p. 163). VR systems offer interactive virtual fly-through facilities with highly photo-realistic content (Duchaineau et al., 1997; Meyer et al., 2001) providing 3D view and seamless spatial zooming in and out functionality. Shortcomings of current spatial-oriented applications are that they cannot model effectively dynamic spatial processes. In general, they lack representing dynamics and their concepts of landscape are static (De Vasconcelos et al., 2002; Pang and Shi, 2002; Peuquet and Niu, 1995). Such systems are very much influenced by cartographic concepts using a series of snapshots to record states of spatial aspects (Chen and Jiang, 2000; Chrismann, 1998; Peuquet, 2001). Moreover, GIS applications often are very large systems tending to be monolithic and therefore costly to combine with other systems (Preston et al., 2003). The process-oriented aspects of this envisioned tool may be supported by *Temporal Simulation Systems* (TSS). TSS address topics such as the simulation of dynamic dependencies. They are concerned with "system states, mass balance and conservation of energy, that is, focusing on quantities [...] in time" (Brimicombe, 2003, p. 163). Due to the hierarchical structure of state-of-the-art simulation models, the composition of complex systems is facilitated through the coupling of models (De Vasconcelos et al., 2002; Zeigler, 1976; Zeigler, 1990). This allows the integration of different process models, which may supports the representation of the respective phenomena in a more holistic, cross-scale and therefore scale-sensitive manner (Steyaert, 1993). A drawback is that in general the spatial dimension is neglected, treated implicitly (Brimicombe, 2003) or only poorly represented, for example through parameterization of spatial properties. Generally speaking, using only one of these tools to study spatiotemporal, cross-scale processes is likely to lead to isolated views and limited understanding of processes, since only the aspect of the process the tool is designed for, is represented sufficiently.

A way out

Taking into account the in crucial aspects complementary strengths and weaknesses of the tools in the respective domains — TSS, VR, and GIS —, it seems a promising approach to combine them into a common framework. This thesis is part of a project which aims at bringing together the three "worlds" TSS, VR, and GIS. By combining applications of the three domains and exploiting the particular strengths of each application the handling of spatiotemporal and cross-scale processes forming the landscape can be improved. The beneficial combination and potential synergies of combinations of TSS and GIS (Bernard and Krueger, 2000; Brimicombe, 2003; Fedra, 1993; Fedra, 1996; Goodchild, 1996; Raper and Livingstone, 1995; Vckovski, 1998) and of VR and GIS (Huang et al., 2001; Lindstrom et al., 1997; Pajarola and Widmayer, 2001) are widely acknowledged. Considerabley less work was dedicated to combine TSS and VR and to the combination of all three domains (Wang, 2004). In the following the term combination is used to refer to the approaches of using software applications in conjunction, for example coupling or integrating.

Combining TSS, VR, and GIS promises advantages through cross-fertilization and mutual support, but it is neither conceptually nor technologically straightforward. One underlying core problem is the differing data models used in GIS and TSS (Aspinall and Pearson, 2000; Bennett, 1997; Fedra, 1993; Fedra, 1996). In GIS, the data model is centered on digital representations of geographical space, the objects located there, and their relationships to one another. The focus is on location, form, dimension, and topology. TSS data models are designed to model processes, their states and throughputs of quantities. GIS is designed to model static representations, TSS is specialized in modeling dynamic systems. The differing emphases, from which a combination of applications could profit, result almost necessarily also in different conceptual and technological structures (Brimicombe, 2003). Nevertheless, there are numerous examples within the literature of combining TSS and GIS (Aspinall and Pearson, 2000; Bernard and Krueger, 2000; De Vasconcelos et al., 2002; Raper and Livingstone, 1995) and of VR and GIS (Huang et al., 2001; Pajarola and Widmayer, 2001) which result in considerable gains in functionality. In the field of planing and scenario generation examples of combining TSS and VR can be found (Camara et al., 1998; Wang, 2004). The different approaches rely on distinct combination strategies from loose coupling of applications, where mainly the data exchange is automated to integration of applications, where the functionality of one tool is integrated within the other (Wittmann, 2000). In general it is crucial to be aware of the fact that GIS are not simply sources of spatial data, but provide functionalities for integrating, handling, analyzing and manipulating spatial data (Brimicombe, 2003).

The IPODLAS project

The thesis is part of a project called *IPODLAS*, which stands for **i**nteractive, **p**rocess **o**riented, **d**ynamic **l**andscape **a**nalysis and **s**imulation. The aim of the IPODLAS project is to combine functionalities from the domains TSS, VR, and GIS, which allows the holistic handling of spatiotemporal and cross-scale processes. The IPODLAS project focuses on identifying concepts and models which allow to meet this aim and which do not limit the scope of development to only what is possible with the contemporary existing functionality (Allgöwer et al., 2001).

The goal of the thesis is to develop the *IPODLAS framework* to combine functionality of the three domains, TSS, VR, and GIS. The framework consists of the IPODLAS approach and the IPODLAS system. The *IPODLAS approach* is a methodology that specifies concepts and approaches to support the development of a system that can satisfy the requirements of the IPODLAS project. Starting with the analysis of the system's requirements and extending further during the design phase of the approach, the focus is on determining the necessary characteristics and functionalities a system such as IPODLAS must comprise. The *IPODLAS system* combines applications of the three domains — TSS, VR, and GIS — as subsystems to facilitate the seamless usage of their functionality, data, and models. To achieve this, concepts and interfaces are to specify providing support for information exchange between the different types of subsystems, i.e. the TSS, VR, and GIS subsystem.

Within the IPODLAS project, three subprojects are conducted. Roughly speaking each domain (i.e. TSS, VR, and GIS) is taking care of one participating subsystem; this thesis is focused on the GIS subsystem. Due to the combined usage of applications from the three domains the three subprojects constitutes a considerable source of data, which was used in the IPODLAS project and partially also exploited in the thesis. In particular,

LBM data used in the thesis originates from TSS model runs executed by Brownyn Price. LBM dynamics pictures and WLF spread images presented in the thesis represent screenshots of visualizations generated at the VR application by Y. Wu. Spatial functionality requirements requested by the IPODLAS project was contributed by the author applying the GIS application. The IPODLAS project originated from the research proposal 'Knowledge Based Dynamic Landscape Analysis and Simulation for Alpine Environments'[1] which is part of the module 5 'Virtual Representation'of the National Research project NRP 48 'Landscape and Habitats in the Alps'[2] of the Swiss National Science Foundation[3].

Research objectives and questions

Within the IPODLAS project this thesis focuses on the design activity to develop a system that can enhance the ability of users to cope with spatiotemporal, cross-scale processes. As stated above, the key idea of the thesis is to use TSS, VR, and GIS together to exploit the — to a considerable degree — complementary strengths and to avoid the weaknesses. The overall goal is to provide an IPODLAS system from where data, functionality, and models of the different domains can be accessed seamlessly.

A major objective of the thesis is the specification of a suitable workflow of getting from direct user requirement to more abstract principles, which are important for specification of the software architecture. The workflow development consists of the collection of functional requirements of users in use cases descriptions and functionality listings. These documents provide guidelines for the incremental and iterative design of the software architecture, which supports the requirements. Additionally, the influence of other requirements and constraints is described, such as legacy systems and nonfunctional requirements. Taking primarily a GIS perspective, an second objective is to analyze wheter standard GIS functionality provides all major functionality, which is required for the application of a GIS in combination with subsystems from the domain TSS and VR when coping with spatiotemporal, cross-scale processes. The third objective is the specification of a suitable software architecture, which results from the described development methodology. The software architecture integrates all subsystems supporting information exchange and coordination between the subsystems.

This GIS-centered approach focus of this thesis influences the selection of use cases and the derivation of requirements towards the software architecture; the spatial, GIS-based perspective may be emphasized. Focusing on some pivotal problems described in the research objectives the following research questions are addressed in the thesis:

1. What is the appropriate development methodology to gather the full range of user requirements and system constraints for the development of a system such as IPODLAS?
2. Is the standard GIS functionality sufficient to support the requirements of a system such as IPODLAS?
3. What are suitable concepts and architectures for a software system to meet the goals of the IPODLAS project, which are to develop a system combining the three

[1] http://bscw.geo.unizh.ch/bscw/bscw.cgi/d77727/nfp48_scient_final.pdf (accessed February 8, 2006)
[2] Nationales Forschungsprojekt NFP48 Landschaften und Lebensräume der Alpen (http://www.nfp48.ch/, accessed May 1, 2005)
[3] http://www.snf.ch/default_en.asp (accessed October 10, 2005)

domains, GIS, VR, and TSS to facilitate the joint seamless usage of functionality, data, and models?

Research approach

The pivotal concept of the IPODLAS framework is handling spatiotemporal, cross-scale processes through exploitation of strengths and avoiding weaknesses of functionality of subsystems of the domains TSS, VR, and GIS. The successfull combination of functionality of the three domains is faced with challenges related to the inherently complex and heterogeneous environment not only on the scientific level, but also to a considerable amount through methodological and technical aspects.

To derive user requirements in a transparent and reproducible way and subsequently develop concepts and software, case studies has been defined providing data and simulation models at several scales. In this manner, cross-scale processes can be represented in an appropriate way. To cover a broad range of requirements of landscape analysis applications, representative case studies have been chosen within differing realms. A standard software engineering framework has been used to specify several IPODLAS system usage scenarios within the case studies captured in use cases. The use cases define the functional requirements of the IPODLAS system. In addition, the scenarios also act as a test bed for the implementation. Prototyping is used to develop and test incremental states and different aspects of the system in the process of software development. In the IPODLAS project several prototypes have been developed examining one or several aspects of software engineering such as synchronization of subsystems, integration strategies, and communication between subsystems.

One goal of the IPODLAS project is to offer landscape visualization, analysis, and modeling functionality in a transparent way to the user, but not to reinvent TSS, VR, or GIS functionality (Allgöwer et al., 2001). The user may access the data and functionality of the respective subsystems independent of the location and type of subsystem. Thus the IPODLAS system must specify interfaces and communication concepts that are independent of the environment and the respective subsystem. They should be applicable locally or over network and for different types of subsystems. To obtain broadly applicable concepts and the resultant software architecture, in general standard software techniques are applied to combine TSS, VR and GIS in a platform independent and interoperable manner. On the design level, a modular system aims for minimal interfaces with limited interdependencies between the subsystems (Ghezzi et al., 2003). The use of interoperability approaches and standard network software technology allows definition and implementation of a software system connecting applications running on different platforms and supporting their interoperation.

Structure of the thesis

The thesis is divided into three parts: a theoretical part I, a experimental part II, and a concluding part III. Part I consists of the chapters 1 — this introduction — to 4. Methods used to develop the IPODLAS framework, ranging from rather general software development principles to methods applied to implement the functional requirements, are detailed in chapter 2. In chapter 3 the state of the art of the — for this thesis — most relevant aspects of the three domains TSS, VR, and GIS is presented. Chapter 4 gathers different arguments discovered in the chapters 2 and 3 and condenses them to the motivation and research approach of the IPODLAS framework. Part II comprises the

chapters 5 to 9, where the achieved results are presented and discussed. Chapter 5 presents the development approach applied to develop the IPODLAS framework. It gives an overview of the layout of the case studies and the usage of the use cases. In a second part, the process of deriving requirements, which influences and constrains the software architecture are outlined via concrete examples. Research conducted in the IPODLAS project and implementation of important aspects of the IPODLAS system in various prototypes is described in chapter 6. The software architecture and communication and synchronization concepts of the most advanced prototype is exposed in chapter 7. Part III encapsulate the evaluating chapters of the thesis: the dicussion in chapter 8 and the conclusions and the outlook in chapter 9.

2 Fundamentals of software technology

This chapter explains fundamentals from the field of software technology that have been applied in this thesis. Section 2.1 describes the software development process used to develop the IPODLAS system[1]. Design principles and techniques of software architecture are detailed in section 2.2. The significance of interoperability and distributed computation are the main issues discussed in section 2.3. Communication, storage, and programming resources are delineated in section 2.4.

2.1 Unified software development process (UP)

One of the most promising current practices in software development is the *Unified Software Development Process* (UP) (Jacobson et al., 1999). The goal of the UP is to transform user requirements into a software system. It relies on three key ideas – use case-driven, architecture-centric, iterative and incremental development. *Use case-driven* means that the development process is guided by the user requirements described within the use cases. *Architecture-centric* refers to the concept that architecture provides a view of the whole design leaving details aside. *Iterative* and *incremental* specify the nature of the workflow dividing the development into *Mini-projects*, which are iterations of the development resulting in an increment of functionality (Jacobson et al., 1999). Figure 2-1 depicts the view of the UP on issues impacting the software architecture.

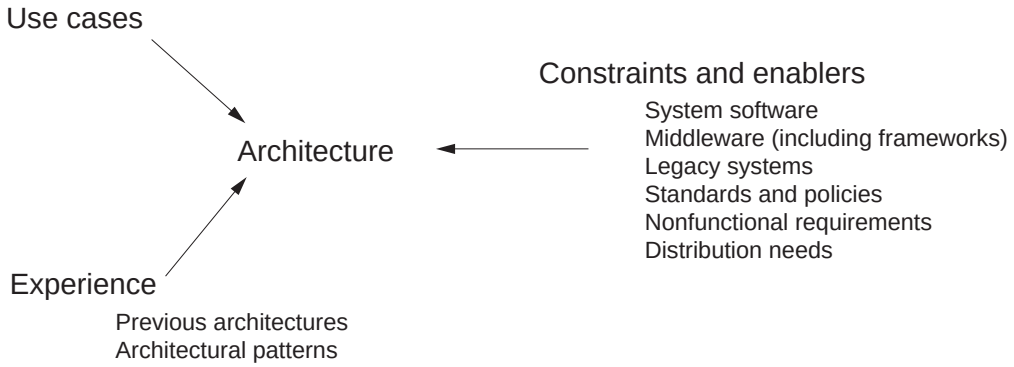

Figure 2-1 Different types of requirements and constraints influencing the software architecture (Jacobson et al., 1999).

Use cases are applied to collect functional requirements. C*onstraints and enablers* summarize conditions, which have to be taken into account when designing software systems. These entail different constraints such as legacy systems[2] (existing applications which are to be incorporated into the new system) or nonfunctional requirements[3] (e.g. reliability or scalability). *Experience* covers the knowledge of the developers (Jacobson et al., 1999).

---

[1] This section is based on section 3.1 and subsection 3.2.1 of Isenegger et al (2005).

[2] Legacy systems refers to already existing software, where usually considerable knowledge is implemented that cannot be replaced easily. On the other hand, due to its traditional software environment, legacy systems are often hard to combine with other software (Ghezzi et al., 2003) in particular when using newer interoperability approaches.

[3] The architecture is not only influenced by the use cases collected in the use case model, but also by nonfunctional, i.e. not use case-specific requirements, such as environmental and implementation constraints, performance, maintainability, etc. (Jacobson et al., 1999).

In the following, the most important issues influencing the software architecture of IPODLAS system are discussed. Case studies are applied to provide real-world data from the test area(s) and simulation models to support the development of IPODLAS system. They help to reduce complexity and act as a test bed for concepts and developed applications. Case studies also assist communication of outcomes to potential end-users (Allgöwer et al., 2001). A *use case* is defined in Jacobson et al. (1999) as: "A use case specifies a sequence of actions […] that the system can perform and that yield an observable result of value to a particular actor". It is a systematic and intuitive means to capture the functional specification of the requirements. A use case usually specifies a particular scenario from within a case study. An common example of a use case is the use of an automatic teller machine (ATM). The withdrawal of money, then, is an example of a use case (cf. Table 2-1) and the use of an ATM by a user the case study (Kotonya and Sommerville, 1998; Windle and Abreo, 2003). All use cases together constitute the *use case model*, which covers the complete functionality of the planned system (cf. Figure 2-2). The definition of the user is done to specify the general intentions of the user, which influence his or her requirements on a software system (Jacobson et al., 1999).

Customer actions	**ATM response**
Customer chooses to withdraw cash	
	ATM asks for amount
Customer enters amount desired	
	ATM checks bank information system, - if sufficient fund is on account - if sufficient fund is in ATM
	ATM returns: - card - cash
Customer takes his/her card and the cash	

Table 2-1 The course of events of the (successful) use case 'withdrawal of cash from an ATM' embedded in the use case model, which is depicted in Figure 2-2. The events are ordered according to their temporal sequence.

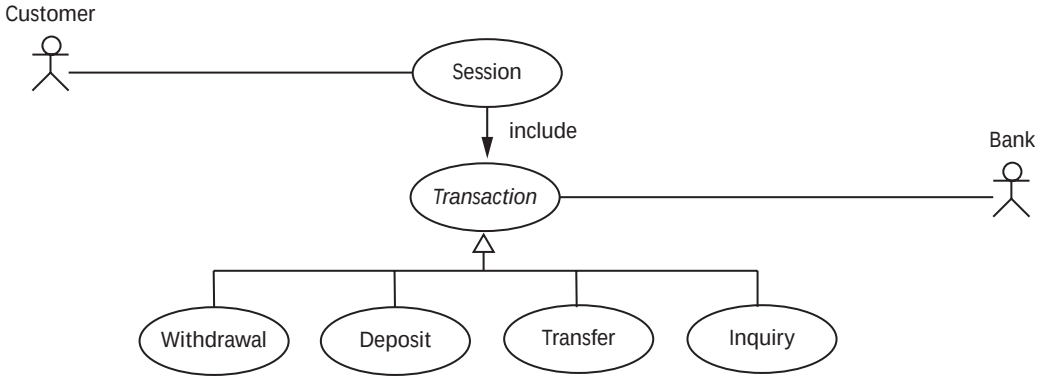

Figure 2-2 Use cases (Session, Transaction, Withdrawal, Deposit, Transfer, and Inquiry) forming the use case model. The use case *Transaction* is an abstract generalization (cf. subsection 2.2.2), which is implemented by one of the use cases Withdrawal, Deposit, Transfer, or Inquiry, after Bjork (2004).

Process of software development

When developing large software projects, it is beneficial to divide the work into Mini-projects. The functionality specified in a use case is implemented in a Mini-project. Each Mini-project is an iteration and results in an increment, for instance in functionality, performance, or user-friendliness. As Figure 2-3 shows, each iteration consists of the steps *Requirements capture*, *Analysis*, *Design*, *Implementation*, and *Test* (Jacobson et al., 1999). A more elaborate description of this process is given by the 'Spiral model of the software process' of Boehm (1988). Among the set of the use cases, the prospective users select the subset, which entails the most important use cases. Jacobson et al. (1999) define the use cases of this subset as *key use cases*: "These key use cases may amount to 5% to 10% of all use cases only, but they are the significant ones, as they constitute the core system functionality". The iterative implementation of the use cases result in an incremental gain in functionality of the prototype, which is developed in parallel to both the use cases and the software architecture. Jacobson et al. (1999) identify several benefits of this controlled iterative process:

- Reduction of risk of failure: Due to the repetitive application of test phases, each Mini-project is validated. In case of failure, only the iteration concerning the current Mini-project must be repeated.
- Repetitive capture of requirements: In large software projects generally not all requirements are identified or fully understood at the beginning or requirements change during the project life cycle. The iterative mode of operation supports refinement of requirements in successive iterations and adaption of the project to changing requirements
- The complex process of software development is broken down to more clear, focussed tasks.

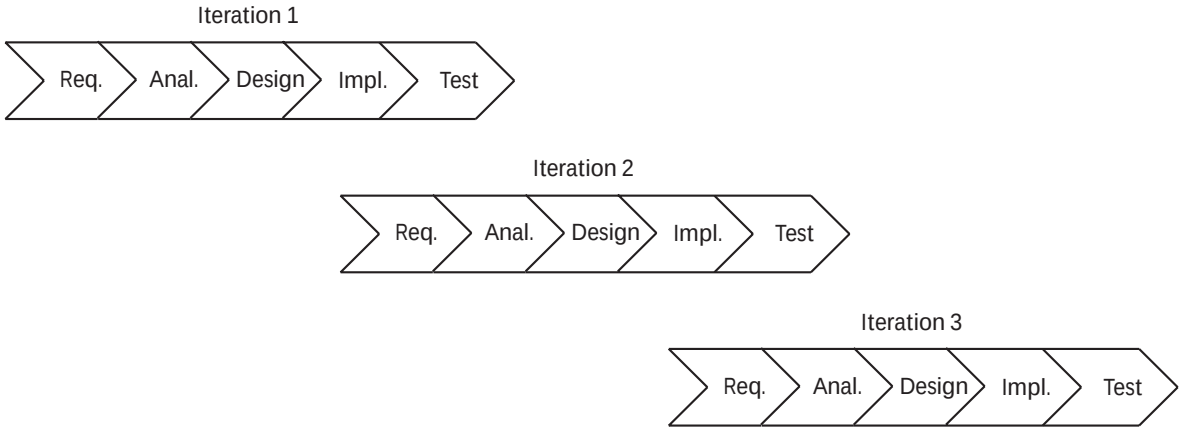

Figure 2-3 Each iteration passes from 'Requirements capture', over 'Analysis', 'Design', and 'Implementation' to 'Test'. The iterations can overlap meaning that one iteration already starts while the last is about to finish (Jacobson et al., 1999).

2.2 Software architecture

IEEE[4] defines *software engineering* in the document 'IEEE 610.12-1990's Standard Glossary of Software Engineering Terminology'[5] on page 67 as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software". Design forms the bridge between requirements towards a system and its implementation, it involves the process of structuring the relations between individual entities. Architectural design decomposes complex systems into subsystems and ensures that subsystems fit together. Software architecture specifies the overall organisation and structure of a system describing its main parts, i.e. its constituent subsystems, and their interactions (Ghezzi et al., 2003). The designer must balance the options, constraints and trade-offs that an architecture exhibits. The architecture provides a means for reasoning about the global properties of a system; it gives a clear perspective on its development (Ghezzi et al., 2003; Jacobson et al., 1999).

2.2.1 Design principles

Design helps to divide a system in subsystems to reduce complexity. This step-wise refinement of the problem is a *top-down* process carried out on iteratively in identified subsystems. Opposed to this, *information hiding* is a *bottom-up* approach, hiding details from closed structures and exhibiting only required information, i.e. the interfaces. The term *Yoyo-design* describes the iterative application of these two antithetic, but complementary principles to gain a holistic view of the software to be developed (Ghezzi et al., 2003).

A software system should satisfy the requirements of its users. Since the requirements of users, the manner of usage and also the users can change over time, the requirements towards a software are likely to be unstable. Hence, software must evolve over time adapting to new requirements. The goal in software engineering is to develop an architecture, which makes the system resilient to change or change tolerant (Jacobson et al., 1999). Important tools in software engineering are programming languages and, in particular, their modularity features. They help to structure and subdivide complex problems into smaller units and they support the separation of specification and implementation (Ghezzi et al., 2003).

Software component and component models

Software components (or just components) are software units which are completely defined through their interfaces. The use of components supports reusability of the programmed functionality and enhances their context-independence (Ghezzi et al., 2003; Szyperski, 1998). Software components should be compatible with other components according to a component model specifying how the components communicate and which common services may be used. Applying components of a system according to standardized component models fosters interoperability of the components and thus of the system (Wytzisk, 2003).

---

[4] The Institute of Electrical and Electronics Engineers (IEEE) is an international non-profit, professional organization for the advancement of technology related to electricity and information technology (http://ieeexplore.ieee.org/iel1/2238/4148/00159342.pdf?isnumber=4148&prod=STD&arnumber=159342&arSt=&ared=&arAuthor=, accessed January 08, 2006)

[5] http://www.techstreet.com/cgi-bin/detail?product_id=16465 (accessed January 08, 2006)

Standard architectures

For systems dealing with common problem standard architectures have been developed (cf. Figure 2-4). In the *pipeline architecture* each subsystem accepts the output of the previous susbsystem as input and delivers output to the next subsystem. This architecture is also called pipe-and-filter architecture perceiving each subsystem as filtering the data. The communication is local meaning that only neighboring subsystems communicate with each other. If subsystems must be able to communicate with more than one subsystem, the *blackboard architecture* may be useful. One subsystem is designated as "blackboard" serving as the communication interface between the other subsystems utilized for exchanging information. In an *event-based architecture* the subsystems can create and react to the occurence of events, i.e. on the arrival of a message, which are propagated on a bus connecting the subsystems. Event-based architectures are appropriate when the subsystems wait for input or when no clear client-server relationship exists (Ghezzi et al., 2003).

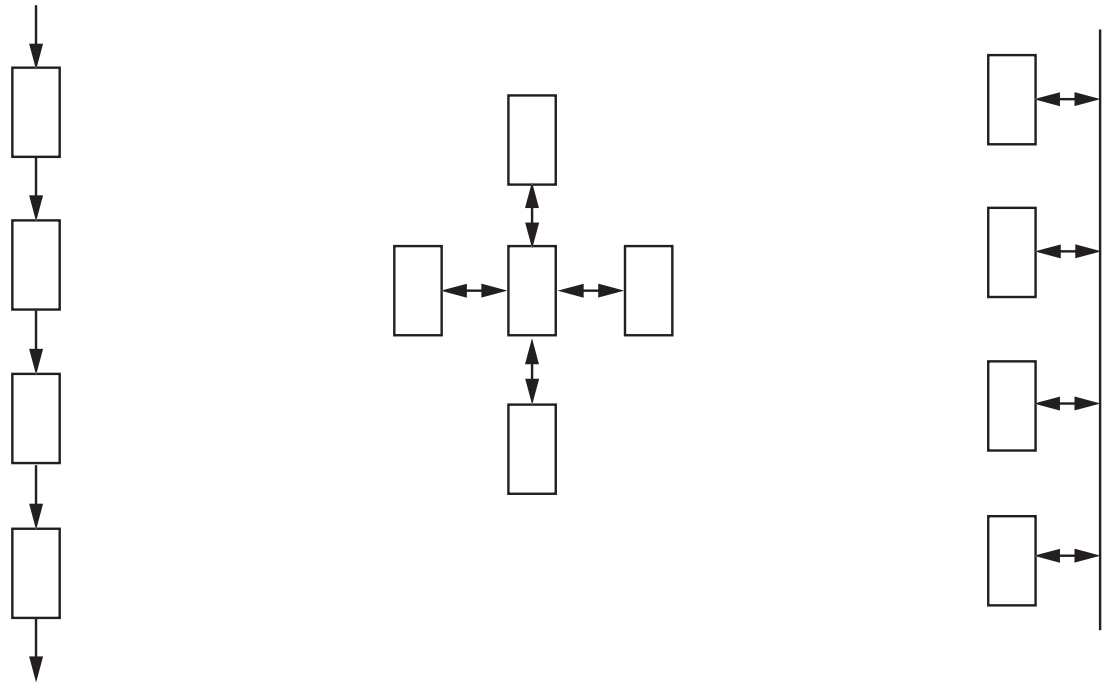

Figure 2-4 The relationship of components in standard architectures, after Ghezzi et al. (2003).

Architecture as framework for component integration

Component-based software development can be seen focusing on two aspects. On the one hand, the available components which provide the required functionality have to be identified. On the other hand, the appropriate software architecture to integrate the selected components must be developped. These two processes affcet each other mutually: the components to be integrated constrain the architecture and the selected architecture constrains the components that can be integrated. As the domains mature, more components underlying the major standards are available. In this case, the software architecture acts as a framework for integration of a set of components; it defines the way the components are arranged and connected (Ghezzi et al., 2003).

With the proliferation of networks and distributed environments such as CORBA, DCOM, and EJB (cf. subsection 2.3.2), the need for standard architectures of distributed systems has grown. The *client-server* architecture is a standard *two-tiered* architecture (cf. Figure 2-5 a), which exhibits two levels of components: the client level and the server level. Components on the client level request services of components on the server level. The *World Wide Web* (WWW) employs this structure; the web browser on the client's computer requests a web page from a web server (usually residing on a remote sever), the web server processes the requests and sends the requested web page to the client. The browser then gets the page and displays it. In more complex situations, an additional layer of functionality can be distinguished forming a *three-tiered* architecture. As in the client-server approach, the client tier sends a request for a service, for example, a database query. This request is received by the second tier or middle-tier (also called middleware or *business layer*), which analyzes and interprets the request. The application tier, in the example of the Figure 2-5 b the database, then finally receives the request sent from the second tier and performs the requested service. Commonly used specific services can be isolated in *application servers* providing a single application, for example a mail service provided by a mail server. Typical functionality of application servers are integration of databases and legacy systems (Feiler, 2000). Application servers may be viewed as large-grained components integrated in distributed architecture. The second-tier can be multi-tiered itself, then the overall architecture is denoted as *n-tiered* (Ghezzi et al., 2003).

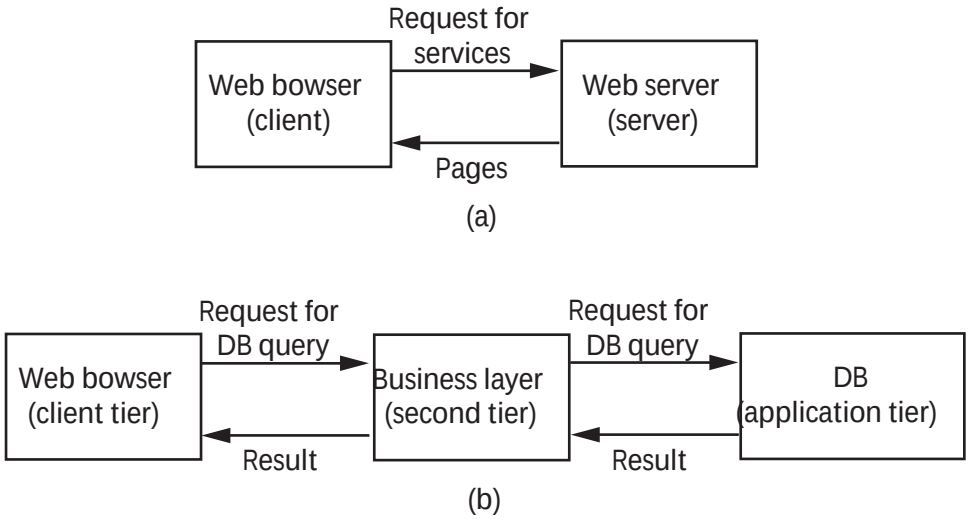

Figure 2-5 a) Two- and (b) three-tiered architectures, after Ghezzi et al. (2003).

2.2.2 Modularity and the object-oriented paradigm

Module, design, and information hiding

A module is a well-defined component of a software system, it encloses certain pieces of software and therefore separates them from each other. Often a module can be considered as provider of a computational ressource or service. Modularization is a result of two complementary aspects of design applying the design principles described in subsection 2.2.1. The architectural or *high-level design* defines the overall structure of the architecture in terms of the interfaces and their relationsships among modules. The *low-level design* focusses

on the structures within a module that are hidden from the other modules (information hiding). Decomposing large systems into simpler pieces helps to break down the complexity, which in particular enhances reusability of the modules through isolating errors and changes to a few locations within the system. Modularity can be increased if the modules exhibit *high cohesion*, which means that all elements of a module are related to each other, that they are grouped together for a logical reason, and that they cooperate to achieve a common goal. In addition, modularity is enhanced, when modules show *low coupling*[6], which represents the interdependence between two modules (cf. Figure 2-6) (Ghezzi et al., 2003).

To support the interchangeability of the subsystems, the whole system is divided into subsystems, i.e. the design is modular. The resulting decrease of the communication load between subsystems limits the dependencies between the individual subsystems and therefore supports interchangeability of the subsystems (Bernard and Krueger, 2000).

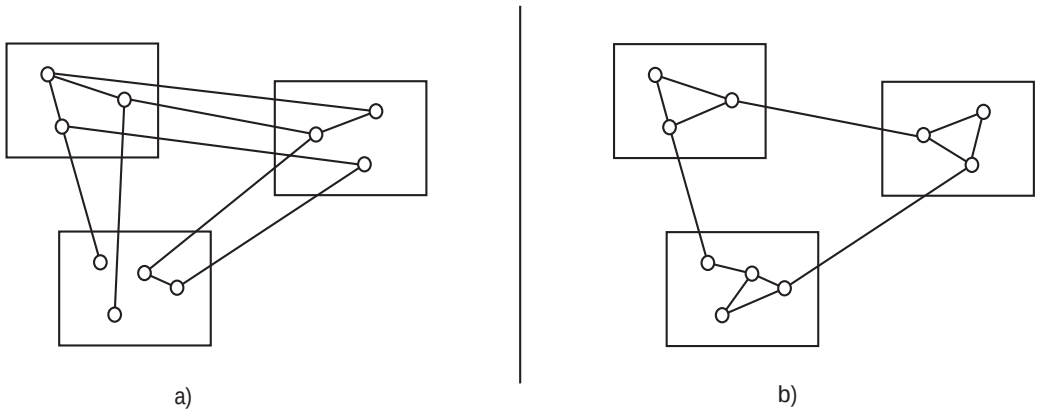

Figure 2-6 Graphical description of cohesion and coupling. Figure a) shows a highly coupled structure with low cohesion, whereas b) illustrates a structure with high cohesion and low coupling, after Ghezzi et al. (2003).

Interfaces

Interfaces consist of a set of services a module provides to the client. As described in section 2.2, the module exports these services, and the client imports them. An interface can be considered as a contract between the provider of a service and the clients. The deliverables of the service are detailed exactly, but the concrete implementation of the service is hidden from the client. Hence, the module implementation can be changed as long as the interfaces remain stable. Precise specification of the interfaces and the separation of specification and implementation enhances the reusability of modules (Ghezzi et al., 2003).

Modules and Object-Orientation

If a module consists of a data structure that can store permanent values, it exhibits a *state*. These modules are called *abstract objects*. An *abstract data-type* module consists of data and provides operations for manipulation. Those operations have exclusive access to the data encapsulated in the module (Ghezzi et al., 2003). *Object-oriented* (OO) design is based on

[6] The principle denoted here as *low coupling* describes the property of exhibiting a modular structure on a more abstract level than *loosely coupled* defined in the subsection 3.4.1.

abstract data types that are known as *classes*. Classes may consist of *attributes* and *methods* that are used to manipulate the instantiazations of the classes called *instances* or *objects*. Attributes which values are shared by all instances of a class are known as *class attributes*. OO design supports the building of hierarchies organizing the classes through *generalization-specialization* relations; the *inheritance* mechanism is used to specify generalization-specialization. Class B inherits from class A, if B is derived from A meaning that A is the *parent* or *base* class of B and B is the *subclass* or *child*. In other words, B specializes A and A generalizes B. From a software architectural point of view, generalization-specialization supports the reusability of components: general properties are maintained in base classes from where specialized *subclasses* appropriate to the respective situation can be derived. Since a derived class inherits all attributes and methods of its base class, a derived class is allowed to appear wherever the base class can appear. This is known as *substitutability*. The subclass is even allowed to redefine inherited methods. The substitutability principle allows that an instance X of a base class can be bound to any instance of its derived classes, which is called *polymorphism*, and the methods that are invoked depend on the derived object that is bound at runtime to X. This is known as *dynamic binding* (Ghezzi et al., 2003). Other relationships in OO design are *association*, which is a set of connections among two or more objects, and *aggregation*, which specifies a whole-part relationship between the aggregate (the whole) and a component part (the part) (Jacobson et al., 1999). The application of the object-oriented paradigm fosters the concepts of information hiding and polymorphism, which supports building of hierarchies, structuring of complex systems and the development of interoperable applications (Bernard and Krueger, 2000).

2.3 Interoperability

To support the development of interoperable software systems, it is crucial to consider the relevant standards and to specify a suitable software architecture. Specifying standards means to establish a common perception and thus a common understanding of the phenomena of interest (Wytzisk, 2003).

Interoperability and standards

IEEE defines *interoperability* in the document 'IEEE 610.12-1990's Standard Glossary of Software Engineering Terminology'[7] on page 42 as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged". In this thesis interoperability is used in a restricted manner. Due to lacking cross-domain interoperability initiatives specifying common interfaces, e.g. common application programming interfaces (API), interoperability is applied on the level of data exchange formats.

The consideration of relevant software standards supports the interoperability of subsystems of a software system with each other as well as with compatible legacy systems. A *standard* is a policy to achieve consistent communication between interacting parties. Common examples for standardization organisations are IEEE, W3C[8], OMG[9], ISO[10] and

[7] http://www.techstreet.com/cgi-bin/detail?product_id=16465 (accessed January 08, 2006)

[8] The World Wide Web Consortium (W3C) is an international consortium developing interoperable technologies such as specifications, guidelines, software, and tools (http://www.w3.org/, accessed January 08, 2006).

[9] Object Management Group (OMG) is a consortium that produces and maintains computer industry standards for interoperable applications. Recently, OMG addresses issues such as modeling (programs,

OGC[11] (ISO/TC 211 and OGC are described in subsection 3.3.2). Definition of common communication interfaces involves on one hand defining which information is exchanged and how it is encoded and on the other hand defining how the information is transferred. Besides syntactic definitions of interfaces such as protocols and encoding, also an agreement about the semantics of the exchanged information is also required (Wytzisk, 2003).

Applying standard communication protocols on the design of interfaces ensures compliance of other components and therefore the interchangeability of components and augments the stability of the interfaces (Wytzisk, 2003). Crosswell (2000) identifies three key benefits of applying standards: portability, interoperability, and maintainability. Standards enhance the portability of a component, i.e. the effort required to adapt the component to a new software environment and standards support a common agreement between participating components about the way of exchanging information, that is interoperability. Software systems compliant to certain standards exhibit increased maintainability compared to proprietary solutions.

Data exchange and conversion

Applications, in particularly from different domains, may require their data to be in a certain format; data that is used by several subsystems must usually be converted. Moreover, due to the different domains of applications the semantics of their data structures most probably do not match (Vckovski, 1998). When composing a system from several subsystems, data exchange is a crucial issue. If only a small number of subsystems are to be combined, individual conversion of formats and partially also semantics between subsystems is a suitable and efficient way of exchanging information. A solution heading for a generic and extensible system encourages the usage of common generic data exchange formats (Vckovski, 1998). Another advantage of using a common exchange format for a system such as IPODLAS is its use as interface between the individual subsystems. This enhances the interoperability by providing a clearly defined data exchange interface, which a subsystem must support when added to the IPODLAS system.

2.3.1 The eXtensible Markup Language (XML) family[12]

Semi-structured data (SSD) is between virtually unstructured data such as raw text and highly structured data, often maintained in rigid structures in databases. SSD is not completely structured nor is the structure guaranteed to be static. Compared to highly structured data the main differences of SSD are missing, additional, or multiple attributes or values, which cannot be constrained to regular or fixed schema used in databases (Shrestha, 2004).

---

systems, and business processes) as well as model-based standards (http://www.omg.org/, accessed January 08, 2006).

[10] The International Organization for Standardization (ISO) is an international standard-setting body composed of representatives from national standards bodies.
(http://www.iso.org/iso/en/ISOOnline.frontpage, accessed January 18, 2006)

[11] The Open Geospatial Consortium, or OGC, is an international standards organization developing and implementing standards for geospatial content and services, GIS data processing and exchange (http://www.opengeospatial.org/, accessed February 28, 2006).

[12] In this subsection, XML and XML-based languages applied in this thesis are described.

eXtensible Markup Language[13] (XML) (Harold and Means, 2004) supports an unambiguous representation for SSD (Shrestha, 2004; Wirz, 2001). XML is a cross-platform-, software- and hardware independent language for transmitting information. It can be applied to structure, store, and exchange information and XML is human- and machine-readable. In addition, XML provides with the *Document Type Definition* (DTD) or the *XML Schema* a self-descriptive mechanism (this feature is characteristic for SSDs). A DTD or an XML Schema is used to specify the structures of an associated *XML Document*, they define the elements and attributes, the number of child elements and the order of appearance, and other structures (Shrestha, 2004). Since the DTD and the XML Schema is extensible, new data structures, i.e. new elements can be added (Hoheisel, 2002; Jones and Drake, 2002; Wirz, 2001). The DTD mechanism suffers from some drawbacks compared to the XML Schema: it is composed of non-XML syntax, only one DTD can be referenced from a XML document, and it provides weak support for validity checks for different data types. *XML namesspace* is a mechanism to avoid conflicts that arise when two or more elements that have the same name appear in the same XML document. XML namespace disambiguates elements with the same name by assigning an additional (unique) identifier to the element (Jones and Drake, 2002; Shrestha, 2004).

XML encodings are a special case of text-based encoding. XML Documents exhibit a hierarchical structure with nested entities. It is a meta markup-language allowing users to define their own document structure, i.e. to introduce their own entities designated with their own markup elements, the tags (Neumann and Eckstein, 2002; Shrestha, 2004). An XML Document contains text, data, and markup elements. The markup elements are part of the self-describing nature of XML: they indicate what information is described in the document. XML 1.0 specification provides certain rules that compliant documents must follow. XML Documents are denoted as *well-formed* if they satisfy a certain grammar, e.g. all tags must be closed, the elements of a document must be case-sensitive, tags must not overlap, attribute values must be quoted, and so on. XML Documents that also conform to the elements defined in their specific DTD or XML Schema, are denoted as *valid* (Lake et al., 2004; Shrestha, 2004).

Figure 2-7 shows a simple XML Document called 'greetings.xml' and the associated XML Schema 'greetings.xsd' defining the elements of 'greeting.xml'. XML Documents and XML Schemas should start with the declaration specifying the XML version. The term `xmlns=http://www.geo.unizh.ch/~disen` in `greeting` denotes a namespace; it indicates that the fragment `greeting` and all its descendants belong to the default namespace `http://www.geo.unizh.ch/~disen`. The XML Document is associated with the XML Schema 'greetings.xsd', this is done by adding the fragments starting with `xmlns:xsi` and `xsi:schemaLocation` to the element `greeting`. The sub elements `from`, `to`, and `message` are located within the root element `greeting`. This structure is defined in the XML Schema 'greetings.xsd': the `greeting` element is of type `xs:complexType`, since it contains the sub elements `from`, `to`, `message`, which are of type `xs:element`.

[13] http://www.w3.org/XML/ (accessed January 10, 2006)

greetings.xml

```
<?xml version="1.0"?>

<greeting
xmlns="http://www.geo.unizh.ch/~disen"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geo.unizh.ch/~disen greetings.xsd">

        <from>Me</from>
        <to>World</to>
        <message>Hello, world!</message>
</greeting>
```

greetings.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.geo.unizh.ch/~disen"
xmlns=" http://www.geo.unizh.ch/~disen" elementFormDefault="qualified">

        <xs:element name="greeting">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="to" type="xs:string"/>
                                <xs:element name="from" type="xs:string"/>
                                <xs:element name="message" type="xs:string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Figure 2-7 The XML Document 'greetings.xml' and the associated XML Schema 'greetings.xsd'.

The major disadvantages of XML-type languages are inflated data volume due to additional metadata and use of a text-based format for encoding binary data. In addition, the parsing of the XML data causes the computer performance to become critical. Compressing the data and transferring binary data separated from the format description can produce relief to a certain amount (Hoheisel, 2002).

One of the fundamental principles of XML is the separation of content from the form. Information content is independent of the information presentation, so multiple views of the same data can be provided without changing the content. One mechanism describing the presentation of XML data is the *extensible Stylesheet Language*[14] (XSL) (Cagle et al., 2001). XSL is a declarative language for defining stylesheets. XSL consists of *XSL Transformations*[15] (XSLT), which is a language for transforming XML data to other formats, such as HTML, PDF, SVG. An XSL stylesheet transformer accepts XML data and produces the presentation of the XML data that is defined by the stylesheet (Shrestha, 2004; Tidwell, 2001). XSLT mechanism uses are templates containing transformation rules,

[14] http://www.w3.org/Style/XSL/ (accessed January 09, 2006)
[15] http://www.w3.org/TR/xslt (accessed January 09, 2006)

which are applied to document parts that match the definitions in the template. XSLT shows strong similarities with functional programming languages such as *LISP* (Tidwell, 2001; Wirz, 2001).

The most common application programming interfaces (API) for parsing[16] are the *Document Object Model*[17] (DOM) and the *Simple API for XML*[18] (SAX) (Jones and Drake, 2002; Tidwell, 2001). DOM represents the XML Document as a tree structure in an object-oriented manner starting with the root element, which forms the root node of the DOM. In most parsers using the DOM representation, the whole XML Document must be stored in memory. Thus, DOM is best suited for applications where the whole document may need to be accessed and manipulated in an unpredictable sequence. SAX is an event-based, interactive interface for XML parsers. It supports memory efficient, stream-based parsing, i.e. it handles XML information as uni-directional, single stream of data, hence it does not provide an internal representation of the XML Document in the memory (Jones and Drake, 2002; Tidwell, 2001; Wirz, 2001). The *Scalable Vector Graphics*[19] (SVG) (Eisenberg, 2002) is an XML markup language for encoding 2-D vector graphics, both static and animated. SVG supports three types of graphics: vector shapes consisting of paths (lines and curves) and areas, raster graphics such as the *Graphics Image Format* (GIF) and the *Joint Photographic Expert Group* (JPEG), and text. Free plug-ins supporting SVG are available for many browsers and platforms (Eisenberg, 2002; Lake et al., 2004). Other XML-based languages such as WSDL and SOAP are described in subsection 2.3.2; GML is specifed in subsection 3.3.2.

2.3.2 Distributed computation

Distributed systems consists to a certain degree of independent computers connected by a communication network. High-bandwidths and low-error-rates of networks support the deployment of interoperating software subsystems on distributed computers (Ghezzi et al., 2003).

Concurrent SW

In concurrent software there is no single stream of execution (or *thread of control*), developers must deal with multiple threads of control. Distributed software is an important class of concurrent software running on different computers connected by a communication network, e.g. a local area network (Ghezzi et al., 2003).

Concurrent execution of programs on networked computers can be distinguished in *parallel* and *distributed* computation. The goals are increased performance and the deployment of subprocesses[20] at different locations, i.e. on different computers. A system with several communicating processes is considered to be a parallel system if the latency of communication between the CPUs[21] of the different computers is less than about 100 µs.

[16] Syntax analysis or parsing describes the syntactic structure of the original code and produces diagnostic messages related to the syntactic structure (Ghezzi et al., 2003).
[17] http://www.w3.org/DOM/ (accessed January 09, 2006)
[18] http://www.saxproject.org/ (accessed January 09, 2006)
[19] http://www.w3.org/Graphics/SVG/ (accessed January 12, 2006)
[20] A computer process is an instance of a program being executed by the operating system including all variables and states (Stevens, 1990) (cf. subsection 2.4.1).
[21] CPU: central processing unit or processor is the component in a digital computer that interprets and executes instructions and data contained in software (Rembold and Levi, 1999).

Such latencies are achieved when using homogeneous CPUs spatially not too far away from each other and communication protocols with a small overhead, which are in general proprietary (Wytzisk, 2003).

In distributed computation a complex system is divided into several interoperating subystems, which can be deployed on arbitrary computers. It is a crucial property of distributed systems that events processed in one process must not influence events processed at the same time in other processes. A synchronization algorithm assures that the sequence of events, i.e. the causal cohesion of events processed in different processes can be guaranteed (Wytzisk, 2003). The client-server approach (cf. section 2.2) is a popular type of distributed software. An example is a print server executing print jobs of clients (Ghezzi et al., 2003).

Middleware

The proliferation of networks has resulted in the development of many distributed software solutions. Many of them have to address similar problems such as locating and accessing services and finding and communicating with processes. *ObjectWeb*[22] defines middleware as "In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system". Typical middleware services are *name services* to locate processes or resources within the network and *communication services* supporting the communcation between processes such as remote procedure call or message passing (cf. subsection 2.4.1) (Ghezzi et al., 2003). Systems such as DBMS, web servers, and application servers can be designated as middleware. The example of a CORBA-based communication given in the next subsection 'Distributed component models' is an example for communication using middleware.

Middleware is perceived as the set of all services that form an abstraction layer hiding heterogeneities (for instance by converting data into a common format) coming from different operating systems, different network protocols, etc. Different forms of transparency, that is hiding of certain properties from the user, is supported by the use of middleware. *Access transparency* guarantees that the resource can be accessed in a single, uniform way regardless from where. *Location transparent* resources hide their physical location, the user must not be aware of where the resource is located. A resource is *replication transparent*, if it can be replicated among different locations, but they appear to be one resource to the user. *Concurrency transparency* supports and hides the fact that a component may be shared between several users (Manninger et al., 2001).

Distributed component models

With increasingly maturing domain, more and better-suited components are available. The software architecture then forms an integration framework for a set of components, it specifies the way the components are arranged and connected (Ghezzi et al., 2003). Popular examples of distributed component models are CORBA, DCOM, and EJB. The

[22] ObjectWeb (http://consortium.objectweb.org/, accessed February 10, 2006) is an international consortium hosted by INRIA, the French National Institute for Research in Computer Science and Control. ObjectWeb defines middleware on the page with the URL http://middleware.objectweb.org/ (accessed February 10, 2006).

Common Object Request Broker Architecture[23] (CORBA) specified by the Object Management Group (OMG) represents such an integrative framework. CORBA establishes platform and programming language independent communication between objects in distributed systems using the client-server paradigm; its core the *Object Request Broker* (ORB) provides transparent access for clients on remote servers. The servers inform the ORB about their availability and clients query availability of services. Once a client finds out about the availability of a service, it asks the ORB to forward its request to the respective server (cf. Figure 2-8). The ORB delivers the answer of the server back to the requesting client. The *Interface Definition Language*[24] (IDL) provides a set of data types to specify function signature interfaces. IDL is used to define the interfaces provided by the servers, the clients can apply those interfaces to compile and link programs. Legacy systems are integrated in CORBA by writing IDL specifications describing their services and mapping the interfaces into interactions with the legacy software through wrapping programs (Ghezzi et al., 2003; Wytzisk, 2003).

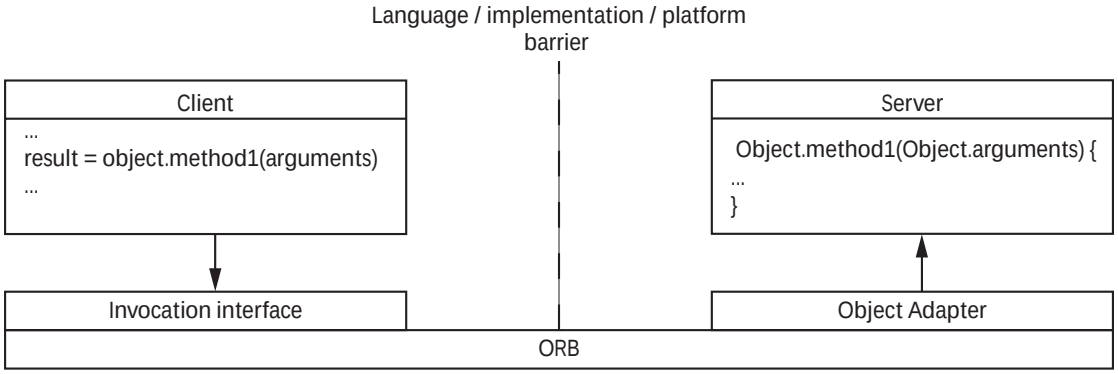

Figure 2-8 Simplified illustration of an ORB-based communication, after Szyperski (2003).

As a competitor to CORBA, Microsoft provides the *Distributed Component Object Model*[25] (DCOM) specifying the communication between components and its clients. DCOM extends the component interaction of *Component Object Model* (COM) with a distributed computing approach. It aims to support communication between distributed objects, which are interoperable and reusable (Wytzisk, 2003). Although COM is a binary standard and can be implemented for any programming language and operating system (Ungerer and Goodchild, 2002), it is mostly used in Microsoft environments. *Entreprise Java Beans*[26] (EJB) is a specification of *Sun Microsystems*[27] for component-oriented, distributed business software. EJB exploits the platform independence of the programming language Java; the communication with other Java applications is established using *Java Remote Method Invocation*[28] (RMI), the Java API for remote procedure calls (cf. subsection 2.4.1) (Wytzisk, 2003).

[23] http://www.corba.org/ and http://www.omg.org/technology/documents/formal/corba_2.htm (accessed January 10, 2006)
[24] http://www.omg.org/cgi-bin/doc?formal/02-06-39 (accessed January 25, 2006)
[25] http://www.microsoft.com/com/default.mspx (accessed January 10, 2006)
[26] http://java.sun.com/products/ejb/index.jsp (accessed January 10, 2006)
[27] http://www.sun.com/ (accessed January 10, 2006)
[28] http://java.sun.com/products/jdk/rmi/ (accessed February 10, 2006)

Web services

The ubiquity of the *World Wide Web* (WWW) promotes web services as another approach to use distributed objects. Web services combine data and functionality, which makes the needed information accessible in the required form. Regarding the heterogeneity of the WWW, web services are only based on interoperable WWW standards such as the *Hypertext Transfer Protocol*[29] (HTTP) and Extensible Markup Language (XML, cf. subsection 2.3.1) (Riedemann and Timm, 2003). Curbera et al. (2001) define the three key issues interoperability, common representation, and usage of standards of the web service framework stating "a web service is a networked application that is able to interact using standard application-to-application web protocols over well defined interfaces, and which is described using a standard functional description language". Other characteristics of the web service framework are a loosely coupled interaction model due to heterogeneity of involved computer environments and the shift of focus from APIs to messages for information exchange. Web services are focused on answers instead of delivering data. Data sets and functionality are split in smaller units and provide the requestor with only what she/he needs (Riedemann and Timm, 2003). The capabilities of a web service are described in the *Web Services Description Language*[30] (WSDL) defining an abstract representation of a service (Riedemann and Timm, 2003). WSDL provides a description of messages exchanged between a web service and its client in an XML Schema. The interface to the service is specified in terms of the set of operations provided, each with specifc input, output, and error messages (Lake et al., 2004).

When using web services as building blocks, the dynamic combination of the single services into a useful *service chain* is often challenging. An approach for automated service chaining is *service trading* illustrated in Figure 2-9. Web services publish descriptions of their capabilities in a *service registry* using *Universal Description, Discovery, and Integration*[31] (UDDI). The registry can be searched by other web services to retrieve a service with required (published) functionality. If a adequate service has been found, the requesting web service gets information how to use the chosen service, i.e. how to *bind* to it (Riedemann and Timm, 2003; Wytzisk, 2003). However, the techniques for exposing (UDDI), describing (WSDL, ISO service metadata) and chaining (ISO service chaining) have deficiencies in handling semantic issues. The description of web services is limited to the syntactic aspect (Riedemann and Timm, 2003).

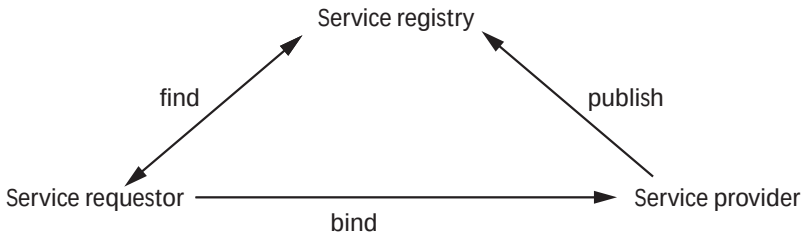

Figure 2-9 Service trading: a service requestor can retrieve published information from the service registry about web services and bind to them according to the obtained specifications, after Wytzisk (2003).

[29] HTTP/1.1 specifications (http://www.w3.org/Protocols/rfc2616/rfc2616.html, accessed January 10, 2006)
[30] WSDL 1.1 specifications (http://www.w3.org/TR/wsdl, accessed January 10, 2006)
[31] http://uddi.org/ (accessed January 10, 2006)

Communication among clients, services, and registries is done via the *Simple Object Access Protocol*[32] (SOAP), which supports the exchange of information over a distributed environment. It is an XML-based protocol enveloping XML messages, which describes the contents of the message and how to process the contents (Englander, 2002). A SOAP message can also transport associated non-XML content, such as binary files (Lake et al., 2004).

Figure 2-10 shows a (hypothetical) example of an integration of two data sets from the Internet using web services. The goal is to determine a hiking route in the Swiss Alps for foreign visitors. This requires topographic data and thematic data (e.g. description of hiking trails, places with facilities for picnic, etc). The topographic information has to be combined with the thematic information, whereas the spatial reference system of thematic data set has to be changed and texts of the thematic data set have to be translated to English. In a first step, the previously identified data sets are investigated using the available metadata and selected using SOAP. In step two, web services are assembled in a service chain, e.g. the web services 'Select area' and 'Transform UTM to Gauss-Krüger', to integrate the two data sets.

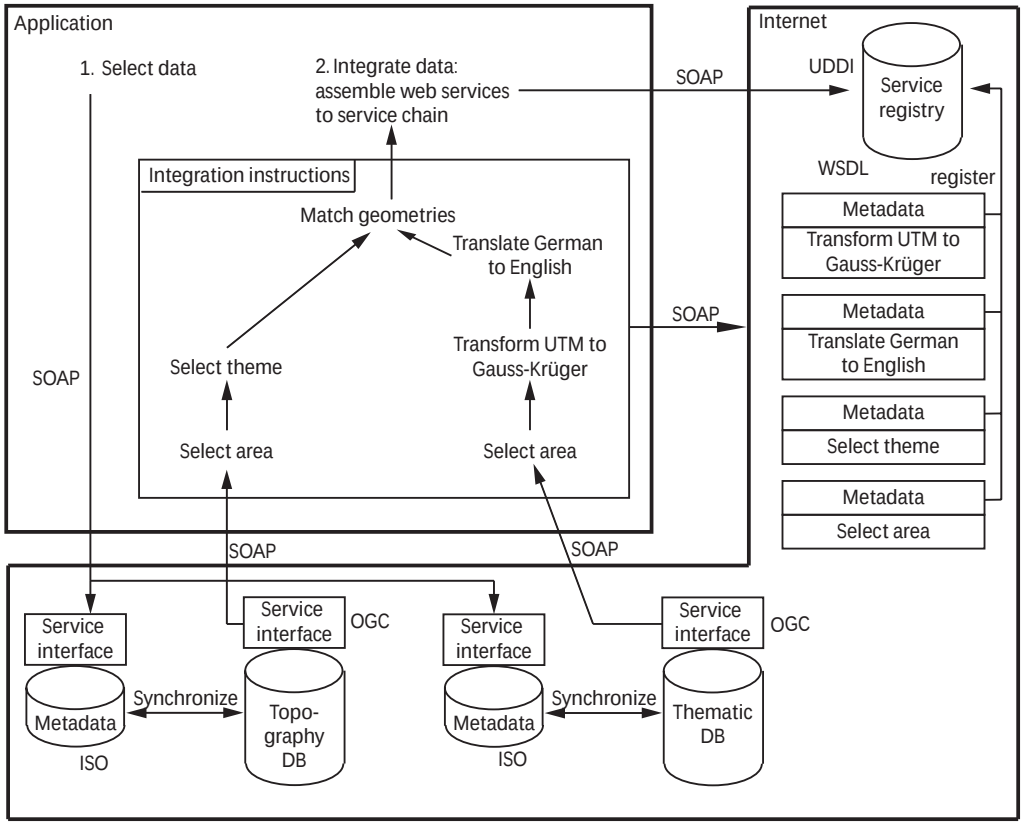

Figure 2-10 The usage of web services for the integration of two data sets. First, the user selects the relevant data from known data sets according to the metadata available. In a second step, web services ('Select area', 'Select theme', 'Transform UTM to Gauss-Krüger', and 'Translate German to English') are discovered in the registry and applied to process the data according to the service chain. SOAP, WSDL, ISO, OGC define the protocols, languages, or specifications used, after Riedemann et al. (2003).

[32] http://www.w3.org/TR/soap12-part0/ (accessed January 11, 2006)

2.4 Communication, storage, and resources

2.4.1 Communication and storage

Inter-process communication (IPC)

A *process* is understood here as an instance of a computer program (a list of instructions implementing an algorithm) executed by the operating system (some operating systems use the term task instead of process). Processes carry state information, have a separate address space, and interact through *inter-process communication* (IPC) provided by the operating system. IPC consists of a set of techniques, such as remote procedure calls (RPC) and message passing (cf. paragraph 'Remote procedure call (RPC) and message passing' in this subsection) , for the exchange of data (Stevens, 1990). IPC can be established by using the *Berkeley* (or *internet*) *socket interface*, which can connect two machines on different file systems (in contrast to the *UNIX socket interface*). In the following the term sockets is applied to denote the Berkeley socket interface. It is a platform independent, connection-oriented protocol (i.e. the connection is always with the same peer) from a client to a server using the hostname and a port number. Inter-process communication is often used for client/server type applications in the TCP/IP domain (Stevens, 1990). Since socket implementations exist for all major platforms and languages, sockets are an often used as a means to connect applications implemented in different languages on heterogeneous platforms.

File-based versus process-based

In a file-based information exchange, an application writes information to a file, while another application reads it from there. In process-based information exchange, the systems communicate via processes and therefore need an interface for receiving and sending information, for example the sockets interface. An advantage of the file-based approach is that writing and reading files to/from a certain location within the storage is a default functionality of all subsystems. The drawbacks of this approach are its restricted speed compared to process-based information exchange and its limited flexibility when using semaphores as synchronization means. The benefits of the process-based approach are its better performance and its greater flexibility. However, these advantages often require higher programming overhead (Wittmann, 2000). File-based information exchange may be adequate in early stages of the development process, since not much programming is normally necessary in establishing file-based communication. File-based information exchange is also preferred if simulation time is much greater than loss of time due to using file-coupling instead of process-based information exchange (Wittmann, 2000).

Remote procedure call (RPC) and message passing

The *remote procedure call* (RPC) allows the calling of processes on different machines which possibly run under different operating systems. *Message passing* is a form of communication for inter-module interaction. Processes having local memory communicate with each other sending and receiving messages; the send mechanism must have a matching receiving mechanism, which can receive the information sent. Variable factors using message passing are the message queue (how many messages can be buffered) and whether the communication is synchronous or asynchronous. RPC and message passing are popular concepts for client-server communication and distributed computing. An important

difference between the two concepts is that RPC inherently implies synchronous communication while message passing can deal with both; a process executing a RPC must wait until the result of the call returns, a process sending a message may continue without waiting for the result of the message (Ghezzi et al., 2003).

Asynchronous communication

Most client-server communication has to deal with multiple network connections simultaneously. For example, a web server may have to answer multiple requests of clients for certain web pages at a time. If the server processes these requests sequentially, all clients except the first have to wait until the first request has been completed. Forking, threading, and asynchronous input/output are mechanisms to handle several network connections at once.

Forking involves *multitasking*, that is the ability to run multiple processes at once (or at least to simulate this behavior). Forking creates a copy of the process; the copy then is the child of the original, which is the parent process (Stevens, 1990). Applying forking, multiple processes can handle multiple requests. In Python (cf. subsection 2.4.3), forking may exhibit varying behavior on the different platforms (Goerzen, 2004). In addition, forking does not scale well, hundreds of connections means hundreds of processes (Rushing, 2005).

Like forking, *threading* permits multiple pieces of code to be executed together. Threading is a way of spliting a process into two or more simultaneously running parts, the threads. Unlike forking, all threads belong to only one process, so changes made in one thread may affects all other threads due to their common address space. While communication between threads is easier than between processes, side effects and the synchronization of the threads can become challenging issues (Goerzen, 2004).

Unlike forking and threading, *asynchronous input/ouput* (asynchronous I/0, also known as *multiplexing*) does not require multiple pieces of code executed together. Instead, a single process observes various connections, switching between and servicing them (cf. Figure 2-11); this is known as *asynchronous communication*. This mode of operation requires mechanisms to handle network connections without blocking other tasks. A *nonblocking socket* connection is such a mechanism, which immediately returns the control back to the operating system after invocation. Furthermore, in many modern operating systems a *polling* mechanism is provided, which checks when the required socket is ready to be used (Goerzen, 2004). Asynchronous communication imposes very little overhead on new connections, this makes it well suited for servers that have to deal with many connections requiring little server-side processing. Also complex synchronization as in threading is not necessary (Goerzen, 2004; Rushing, 2005).

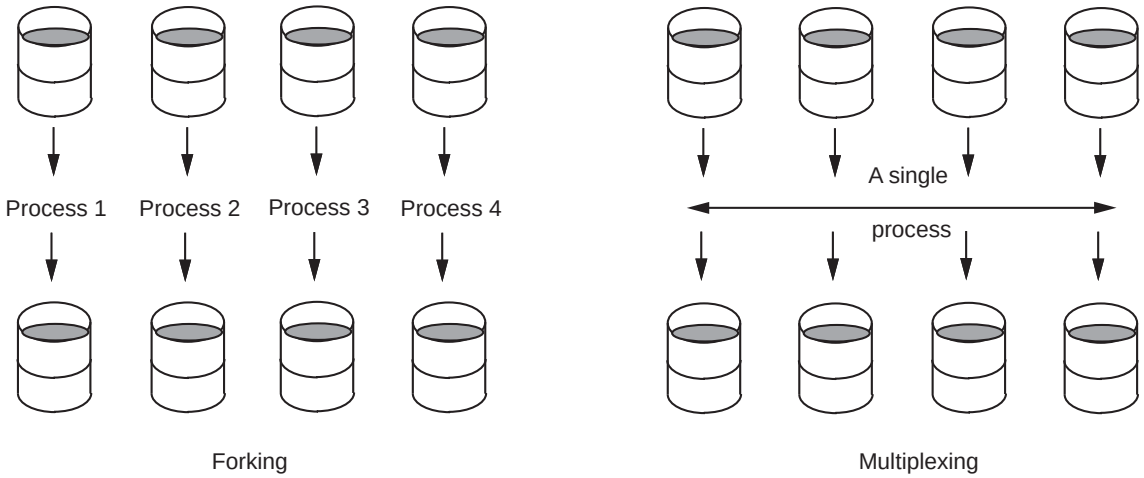

Figure 2-11 Graphical description of asynchronous communication using the metaphor of filling water from the above bucket to the related bucket at the bottom. When using forking one process is responsible for one pair of buckets (in the threading case, it is a thread instead of a process). In the multiplexing case, a single process takes care of all buckets switching between and serving them successively: first some water is filled from the first bucket in the upper line into the first bucket in the lower line, then from the second bucket in the upper line into the second bucket in the lower line and so on, after Rushing (2005).

Storage

The term data storage is used here to encompass both storing files on a fileserver and using a database management system (DBMS). A simple way of storing and exchanging data is via files on a *fileserver*. While access to a file server is a default functionality of most applications, no synchronization of data access is provided. Applying a *DBMS* adds a software layer to facilitate the management of the interactions between program and data (Longley et al., 1999) and provides functionality for data access and storing such as synchronization and fail-safe mechanisms (Worboys, 1999). Data can be stored in *federated data storages*, i.e. local to the source systems. A central server manages remote data access. Due to different data models and formats, a loss of semantics is likely. Using a *data warehouse* provides a homogeneous view of the heterogeneous data stored in a central repository; the data is already integrated (Bergmann et al., 2000; Voisard and Schweppe, 1998). Storing the data in a common format requires the data to be converted in an application-specific structure and format to be processed by the applications, which requires knowledge of the semantics of the data models of the respective systems. This is supported using a canonical data model providing meta data to resolve heterogeneity between data models (Leclercq et al., 1996; Leclercq et al., 1997).

2.4.2 Legacy systems

As described in section 2.1 legacy systems are existing applications to be incorporated into the system to be developed; in the IPODLAS system parts of their functionality are used to satisfy the requirements captured in the use cases and defined in the functionality lists (cf. chapter 5). For the IPODLAS system RAMSES, VTP, and GRASS are considered as legacy systems (cf. subsections 3.1.4, 3.2.4, and 3.3.4). Being typical representatives of applications of their respective domain their integration in the IPODLAS system may bring

up the characteristic benefits and challenges of an integration of these domains. Furthermore, there is in-house knowledge about the chosen applications available to the IPODLAS project team (Isenegger et al., 2005) and all applications are open source software[33], which guarantees the availability, modification, and extensibility of their source code.

2.4.3 Programming language and framework

The most advanced IPODLAS system (cf. Chapter 7) has been developed on the basis of the framework TwistedMatrix, which provided basic communication functionality. Python was applied to establish the communication with TwistedMatrix and other components of the IPODLAS system. In this subsection Python and TwistedMatrix are shortly summarized.

Python

Python[34] (Downey, 2001; Eckel, 2001; Pilgrim, 2004) is a *high-level language* such as Java or C++, and is developed within an open source project. High-level languages must be processed into low-level languages, sometimes referred to as 'machine languages', what slows down the execution of high-level languages. Advantages of high-level languages are, firstly, that they are much easier to program than machine languages and are therefore less error-prone. Secondly, high-level languages are portable.

Python is an *interpreted language* (opposed to e.g. Java and C++, which are compiled languages) providing implementations for the major platforms, which is fully dynamically typed and provides automatic memory management (Downey, 2001). This means that the user need to specify no type definition and no memory when initializing variables. Python supports object-oriented programming, but allows also the application of other paradigms. One of the biggest advantages of Python is its large standard library, which is known as 'batteries included' (Pilgrim, 2004).

The *standard library*[35] offers modules providing functionality suited to many tasks, most of which are compatible across platforms. Examples are the modules 'xml.dom', 'xml.sax', 'xml.parsers', and 'xml.lib' providing functionality for handling XML data (Jones and Drake, 2002). The standard library also provides modules with strong support for Internet-facing applications offering a large number of standard formats and protocols, such as 'socket' supporting the handling of sockets, 'SocketServer', which is a framework for network servers, and 'asyncore' and 'asynchat' for developing asynchronous communication in Python (Goerzen, 2004). Additionally, there exists a wide range of third-party, open source Python modules such as Twisted or PyXML[36]. Many of these third-party modules can be found using the Package Index[37]. In the IPODLAS system, the Python release 2.3 and 2.4 has been used.

---

[33] The code of open source software is available under open source license and can be studied and changed. The Open Source Definition is maintained by the Open Source Initiative (http://opensource.org/docs/definition_plain.php, accessed January 25, 2006).

[34] http://www.python.org/ (accessed January 11, 2006)

[35] All built-in modules can be found on the Python Library Reference, e.g. in the one of Python 2.2.3 (http://www.python.org/doc/2.2.3/lib/lib.html, accessed January 13, 2006)

[36] PyXML is maintained by the *Python XML Special Interest Group* http://www.python.org/sigs/xml-sig/ (accessed January 13, 2006)

[37] http://cheeseshop.python.org/pypi/ (accessed January 13, 2006)

TwistedMatrix

TwistedMatrix[38] (Fettig, 2005) is a Python framework for programming network services and applications using asynchronous I/O. It supports various standards such as TCP, sockets, HTTP, SSH, and FTP. TwistedMatrix supports the usage of delimiters to separate different datasets transmitted through the socket from each other. However, the maximal size of a data chunk exchanged in one part over socket connections established by TwistedMatrix is limited to 16 KB. A crucial concept is *nonblocking asynchronous servers*. Nonblocking means that a call to a server does not wait until the server answers (i.e. block the whole system), but that the call returns immediately. When the server responds, the TwistedMatrix framework gets the answer and then informs the calling application. This paradigm is known as *event-driven* or *event-based* programming. The advantage of this approach is that it allows effectively multitasking without using multiple processes or threads (Goerzen, 2004; Lefkowitz and Shtull-Trauring, 2003). The TwistedMatrix release 2.0.1 was used in the IPODLAS system.

[38] http://twistedmatrix.com/ (accessed January 16, 2006)

3 Fundamentals of Temporal Simulation Systems, Virtual Reality, and Geographic Information Systems

The introduction (chapter 1) addressed potentials and benefits of a combination of the domains TSS, VR, and GIS. In this chapter, aspects of the three domains relevant for combined usage are covered, and advantages and shortcomings of each domain with respect of such a combination are outlined. A key motivation for combining applications from the different domains is their partially complementary strengths and weaknesses. Together, they constitute the building blocks from which the motivation and the wish for a system like IPODLAS emerges. When combining applications, information exchange is a key issue, which is facilitated when all partners involved in software development process conform to certain standards (Wytzisk, 2003). In the following *combine* or *combination* is used to refer to a general joint usage of applications (the terms integrated or coupled denote in some domains a certain way of combining applications). Interoperability refers to the ability of various autonomous systems to exchange meaningful information (cf. section 2.3). A crucial element therein is a contract between interoperating parties defining a set of agreed common features and operations. A contract in this context means standards defining at least data formats, application programming interfaces (API), and communication protocols (Vckovski, 1998). In this chapter relevant standards and their contributions to interoperability are presented for each domain.

In the following, for each domain fundamentals, interoperability approaches, and shortcomings, which are relevant for the development of the IPODLAS framework, are described in the respective section 3.1, 3.2, or 3.3. Each section consists of the subsections 'Basics', 'Interoperability approaches', 'Shortcomings', and 'Legacy system'[1], which describes the legacy system (cf. subsection 2.4.2) of the respective domain used in the IPODLAS system and possible alternative applications. The last section 3.4 assesses and evaluates critical issues of combining TSS, VR, and GIS.

3.1 Temporal Simulation Systems (TSS)

The central topic in TSS applied in environmental modeling is the one of "state, expressed in terms of numbers, mass or energy, of interaction and dynamics" (Fedra, 1993). Modular simulation model supporting hierarchical composition can be combined to simulate complex systems. Thus, different simulation models representing a process on different scales can be combined to accomplish a representation of the process in holistic, cross-scale, and therefore scale-sensitive form (De Vasconcelos et al., 2002; Steyaert, 1993).

3.1.1 Basics

Time

The introduction in chapter 1 illustrates that most environmental processes show spatiotemporal characteristics. This type of information can be considered as a composition of fractional information from space, time, and theme (Peuquet, 1999). Motivated by the spatiotemporal perspective, Frank (1998) distinguishes between *linear* and

[1] The respective subsections 'Legacy system' are based partially on the subsection '3.2.2 Legacy systems' of Isenegger et al. (2005).

cyclic time; he proposes a taxonomy of time exhibiting the categories direction and scale (cf. Table 3-1). Time can be seen in a linear context, where events are ordered according to the time they occur. Perceiving time as cyclical phenomenon (e.g. time classified as time of day), events can occur periodically. Both time perceptions can be scaled ordinally (e.g. when the order of events is known, but not the time) or metrically. Metrical time scale can be continuous or divided using a discrete time unit.

Direction	**Scale**	
Linear	Ordinal	
	Metric	Discrete
		Continuous
Cyclic	Ordinal	
	Metric	Discrete
		Continuous

Table 3-1 Taxonomy of time, after Frank (1998).

For simulation, Fujimoto (1999) defines different notions of time: the *physical time* defining the real time in the physical system. The *simulation time* is the virtual time, a representation of the physical time within the simulation. The *wallclock time* describes the real time during a simulation run. When the simulation is not controlled by the wallclock time, then *as-fast-as-possible execution* (i.e. *unpaced execution*) takes place. *Real-time execution* is *paced* simulation execution; the wallclock time and the simulation time are incremented synchronously. This is a special case of *scaled real-time execution*, where the advance of the simulation time is in synchrony with the wallclock time scaled by a constant factor (Wytzisk, 2003).

Models and simulation

Simulation is understood here as the application of models to reproduce time dependent processes. The conceptual model abstracts the relevant aspects of a process and describes it in a formal way (cf. Figure 3-1). The implementation of the conceptual model in a piece of software allows the simulation of a process over time on a computer, which may produces results leading to new insights about the modeled process (Wytzisk, 2003). State variables represent the current state of the physical system, changes of the state variables model the evolution of the process.

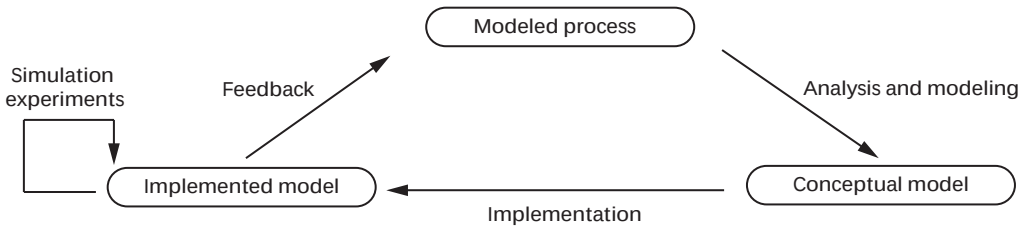

Figure 3-1 Models and simulation (Wytzisk, 2003).

Discrete versus continuous simulation

Simulations can be classified according to their time increment method and their description of the state of the simulated system (cf. Figure 3-2). *Discrete simulations* increment the simulation time in a non-continuous way to the next point in time. The model is usually described by differential algebraic equations. Changes in the state of the

model are atomic and happen only at discrete simulation times. The simulation model describes for all simulation variables how these variables change over time. In *time-stepped* (discrete) simulation, the simulation time is advanced in a constant or variable increment. All events which happen to fall into the considered time period are processed. In the *event driven* (discrete) simulation, the simulation time is always set to the time of the next event to be processed. The Discrete Event System Specification (DEVS) of Zeigler (1976) is a formal description of discrete, event driven systems. Time-stepped systems are described by the Discrete Time System Specification (DTSS) (Zeigler, 1976). *Continuous simulation* increment their simulation time in a continuous way, the states of a model can adopt unlimited numbers of values in each time period. The behavior is typically described by differential equations. In *lumped parameter models* the time is the only independent variable (Cellier, 1991). In *distributed parameter models*, besides the temporal, also the spatial variation is described. Continuous systems can be formally described in a *Differential Equation System Specification* (DESS) (Praehofer, 1992). RAMSES (Fischlin, 1991) (cf. subsection 3.1.4) is the TSS applied in the IPODLAS framework; RAMSES supports working with models compliant to DEVS and DESS (Fischlin et al., 2002).

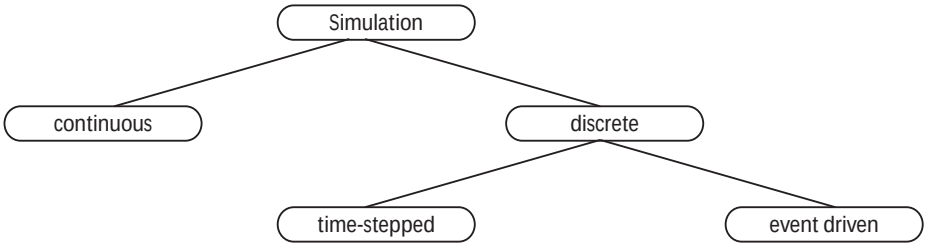

Figure 3-2 Taxonomy of simulation methods, after Fujimoto (1999) and Wytzisk (2003).

Agent-based simulation and cellular automata (CA)

Emergent phenomena can occur in complex real systems. These phenomena cannot be explained by the behavior of a single component of the system, but only emerge through the interaction of system components. One approach to represent emergent phenomena is the use of self-organizing, autonomous entities, which is a property of agent-based simulation (Wytzisk, 2003). Basically, *agents* are autonomous entities which act independently, but may also in concert with other agents. They sense their environment and act on it to fulfill their agenda (Batty and Jiang, 1999). Franklin and Graesser (1997) propose a range of (possible) properties of agents (cf. Table 3-2):

Property	**Meaning**
Reactive	Responds in a timely fashion to changes in the environment
Autonomous	Exercises control over its own actions
Goal-oriented/ proactive / purposeful	Does not simply act in response to the environment
Temporally continuous	Is a continuously running process
Communicative / socially aware	Communicates with other agents
Learning / adaptive	Changes its behavior based on previous experiences
Flexible	Actions are not 'scripted'
Character	Exhibits personality and state

Table 3-2 Properties of agents, after Franklin and Graesser (1997).

Cellular automata (CA) are an agent-based approach defining an explicit representation of space, in GIS usually as cells arranged in a (regular) grid (Batty and Jiang, 1999). The cells of a CA interact with their direct environment. The sequence of states is computed applying rules valid for all cells describing the interdependency of a cell and its neighborhood. A CA defines a tessellation of the space, the neighborhood of a cell, a finite set of states a cell can adopt, and a local function conveying the old state of a cell into the new one (Gerhardt and Schuster, 1995; Shi and Pang, 2000). With the intrinsically spatial reference and the application of transition rules, CA present an approach to integrate (discrete) dynamic models and (raster-based) GIS[2] (Shi and Pang, 2000).

The *spatial representation* of spatial agents (agents which can reason on and handle spatial information), CAs, and raster-based GIS can be seen as a hierarchy of levels starting at the cell as the basic unit of operation. The next level form neighborhoods, which are defined around each cell, usually formed by the cells at the 8 compass directions. The top-level is global, which is sometimes also denoted as world or environment, where actions take place across the entire system. In raster-based GIS an additional intermediate level can be distinguished: cells exhibiting a common property can be seen as the focal level or zone. Figure 3-3 shows the correspondence of concepts in the different approaches. Moving from raster-based GIS to CA modeling, and agent-based simulation the systems get more aware of the temporal dimension and are more decentralised in terms of spatial decision-making (Batty and Jiang, 1999).

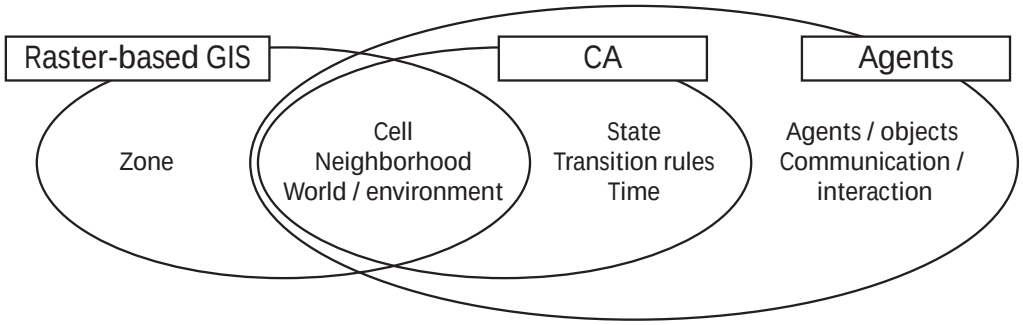

Figure 3-3 Relations between raster-based GIS, CA modeling, and agent-based modeling, after Batty and Jiang (1999).

3.1.2 Interoperability approaches

There are several frameworks for discrete and continuous simulation. The dominant one is the *High Level Architecture for Modeling and Simulation*[3] (HLA) (Wytzisk, 2003), which is a framework for the integration of distributed simulation models. A primary goal of its distributed approach is the increased reusability and scalability due to interchangeable simulation components rather than advances in performance through parallelization. HLA supports distributed simulations where individual simulation components subscribe for a simulation run, receive and deliver information and then unsubscribe. The HLA architecture is based upon three basic specifications (Bernard, 2001; Wytzisk, 2003):

[2] GIS are explained in section 3.3.

[3] The Institute of Electrical and Electronics Engineers, Inc. (IEEE) standardize HLA in the documents 1516-2000, 1516.1-2000, 1516.2-2000, 1516.3-2003. Those are available from http://www.techstreet.com/info/ieee.tmpl (accessed December 22, 2005)

- the HLA *Federation Rules* define the basic rules for the development of simulation models (*Federations)* and their simulation components (*Federates)*. The rules define the specification, generation, and documentation of components as well as the generic aspects of the data exchange and documentation.
- the HLA *object model template* (OMT) is a meta language, which specifies formats of the information exchange between processes
- the HLA *interface specification* defines the *runtime infrastructure* (RTI) describing the interfaces for basic management services realizing distributed simulations. RTI specifies methods, which the federates must provide for the communication between them and with the federation.

RTI is a crucial element of an HLA implementation establishing the communication and synchronization of the individual processes, the federates, of a distributed simulation, the federation (cf. Figure 3-4). All state information about the federation is stored in the single federates. The RTI must not have any knowledge about the semantics of the transmitted information; this fact enhances the universal applicability of the RTI (Wytzisk, 2003).

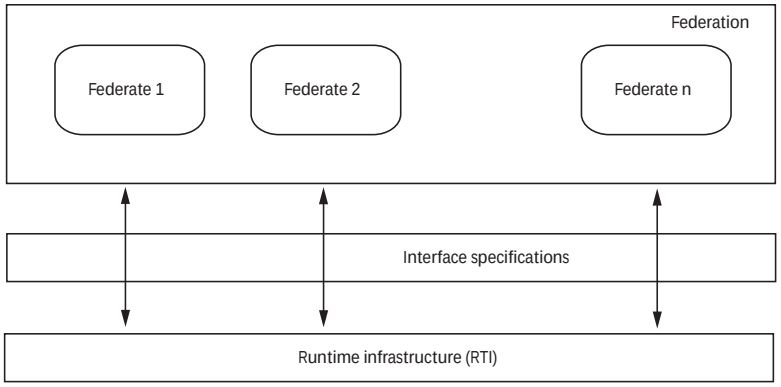

Figure 3-4 RTI-based communication with federates (Wytzisk, 2003).

3.1.3 Shortcomings

Most environmental processes inherently have a spatial dimension and the increasing availability of spatial data creates the desire of using it. However, in general the spatial dimension is either neglected in TSS, treated implicitly (Fedra, 1993), or only poorly represented, for example in lumped-parameter models. TSS often lack the tools for spatial data handling, e.g. spatial analysis and manipulation, which would help making spatial representation and in particular spatial variability explicit for solving environmental problems (Brimicombe, 2003). The standard framework HLA also lacks supporting spatial information and spatial applications, since no standardized object with spatial reference is defined as a federate object model template; specifications for the development of spatiotemporal simulation environments are not available. Moreover, HLA does not support the standardized application of spatial web services as defined in spatial interoperability initiatives such as the Technical Committee 211 of the International Organization for Standardization (ISO/TC 211) or the Open Geospatial Consortium

(OGC)[4] (Schulze et al., 2002; Wytzisk, 2003). Additionally, nor the linkage to spatial databases nor the usage of spatial analysis tools is supported (Bernard and Krueger, 2000).

3.1.4 Legacy system

The *Research Aids for Modeling and Simulation of Environmental Systems*[5] (RAMSES) (Fischlin, 1991; Fischlin et al., 2002) has been evaluated to be one of the most appropriate TSS applications for the needs of the IPODLAS system (Giorgetta, 2002). RAMSES is an interactive modeling and simulation software; the *RAMSES Simulation Server for Unix Workstations*[6] (RASS) (Thöny et al., 1995) is the batch-oriented version of RAMSES. RAMSES can be seen as a toolbox (cf. Figure 3-5) containing tools specified for a particular purpose. One tool is the *Dialog Machine*, which is a library of routines supporting the programming of a user interface. The *Integrative Systems Implementation Software* (ISIS) supports the implementation of complex dynamic systems in a hierarchical manner. *ModelWorks* is the simulation environment of RAMSES, supporting modular modeling.

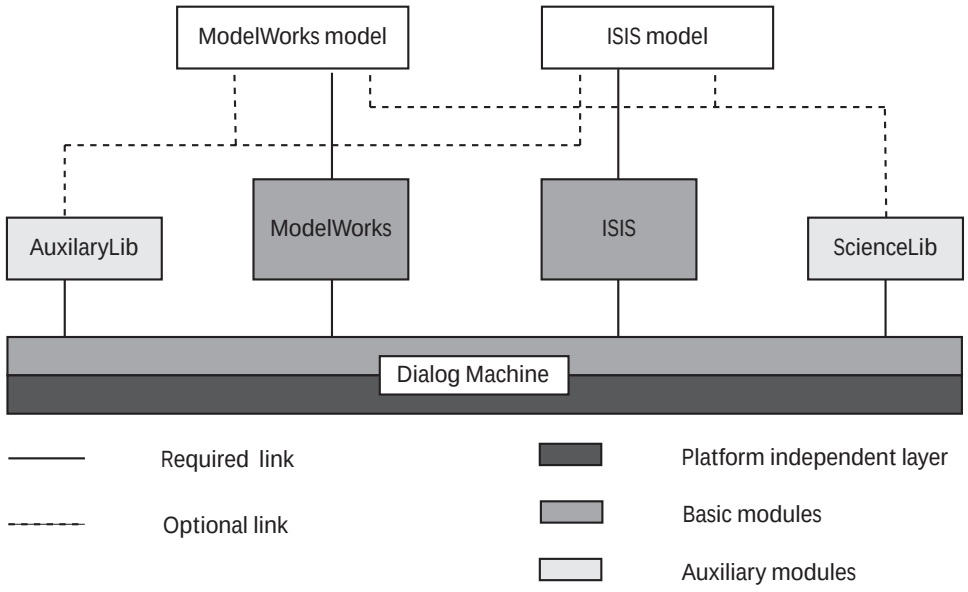

Figure 3-5 Simplified layer model of RAMSES (Fischlin, 1991), after Giorgetta (2002). ModelWorks and ISIS are basic modules. The Auxiliary Library (AuxiliaryLib) supports programming, modeling, simulation, and scientific data analysis, whereas the Science Library (ScienceLib) provides modules for scenario generation, statistical analysis, advanced modeling. All modules consist of many libraries and are shown only in summarized form[7].

ModelWorks allows interactive solving of non-linear differential equations, difference equations, and discrete event systems in any combination using standard model formalisms such as Discrete Event System Specification (DEVS) and Differential Equation System Specification (DESS) (Fischlin, 1991; Fischlin et al., 2002) (cf. section 3.1). RAMSES

[4] Information about ISO/TC 211 and OGC can be found in subsection 3.3.2.
[5] http://www.sysecol.ethz.ch/SimSoftware/RAMSES/ (accessed January 02, 2006)
[6] http://www.sysecol.ethz.ch/SimSoftware/RAMSES/RASS.html (accessed January 11, 2006)
[7] cf. http://www.sysecol.ethz.ch/SimSoftware/RAMSES/What_Is_RAMSES.html (accessed January 17, 2006)

provides a Modula2[8] API and consists of several software layers which offer access to the system's functionality (Fischlin, 1991). For the IPODLAS system the Ramses 3.03 distribution on MacOS X was used.

Possible alternatives to RAMSES are Simulink and Scilab both providing a broad range of simulation functionality, which could be used by the IPODLAS system. Simulink[9] is a supplementary package of the commercial product MATLAB[10] which is a numerical computing environment and programming language. Simulink supports continuous, discrete, and event driven simulation and provides distributions for all major platforms. Providing intuitive graphical tools for modeling hierarchies Simulink benefits from the underlying powerful numerical MATLAB functionality. Due to its widespread use and its large user community Simulink profits by a good user support (Giorgetta, 2002).

Scilab[11] is a numerical computational software package developed by INRIA[12] and ENPC[13] which is distributed freely for the major platforms. Scilab provides broad mathematical functionality, a high-level programming language, and toolboxes such as the system modeler and simulator toolbox Scicos[14]. Supporting continuous and discrete simulation Scicos allows to graphically model and simulate dynamic systems.

3.2 Virtual Reality (VR)

3.2.1 Basics

Simulation of environmental processes, for example the modeling and simulation of insect migration (cf. subsection 4.1.1), often produce spatiotemporal, high-dimensional, and large data sets. To gain better insights and detect patterns, e.g. anomalies and trends, graphical representation of the numerical data is often more adequate than looking at raw or tabular data. Hence, visualization as graphical representation of numerical data is a key element for understanding complex environmental processes (Biegger, 2004; Kraak et al., 1999; Neves et al., 1999; Van Dam et al., 2000).

Applications of VR and foundation

Most VR systems are designed for the display and high-level interaction of the user with the phenomenon of interest, which includes manipulation, navigation, and simple database queries (Williams, 1999). The application of VR systems and immersive virtual reality systems (IVR) is an intuitive way to explore high-dimensional and complex 3-D or even 4-D data and to overcome the limitation of 2-D computer graphics. IVR is a potentially powerful tool for visualizing complex data, it supports working with 3-D information, such as real-time 3-D computer graphics, 3-D sound, and haptic information. These systems provide a natural way of exploring data and interacting with it. Navigation is accomplished with standard input devices, interaction is a means for communication between user and application (Van Dam et al., 2000). VR systems offer interactive virtual fly-through facilities

[8] http://www.modula2.org/ (accessed January 25, 2006)
[9] http://www.mathworks.com/products/simulink/ (accessed September 20, 2006)
[10] http://www.mathworks.com/products/matlab/ (accessed September 20, 2006)
[11] http://www.scilab.org/ (accessed September 20, 2006)
[12] http://www.inria.fr/ (accessed September 20, 2006)
[13] http://www.enpc.fr/ (accessed September 20, 2006)
[14] http://www.scicos.org/

with highly photo-realistic content (Duchaineau et al., 1997; Meyer et al., 2001) providing 3-D view and seamless spatial zooming functionality.

Figure 3-6 shows a 3-D visualization of the forested areas in the Upper Engadine, a valley in the Eastern Swiss Alps, generated by VTP) which is the VR application applied in the IPODLAS framework. The case study Larch Bud Moth (LBM) of the IPODLAS framework used data from LBMs collected in that valley. The 3-D visualization allows a better cognition of the rugged terrain, for example of the slope and aspect of the sites, which denotes, with respect to aspect, more or less homogeneuous forest compartments of the Upper Engadine valley.

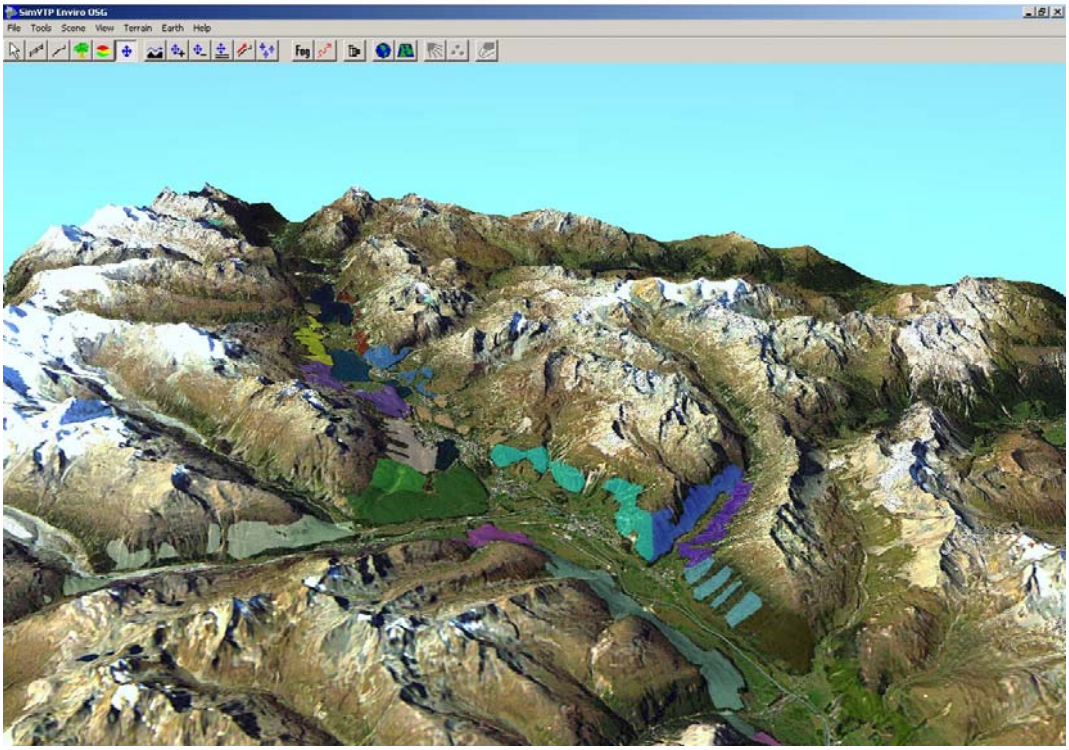

Figure 3-6 3-D visualization of the VR application VTP of the forested areas in the Upper Engadine (Switzerland). The forested areas in the valley are divided in 20 sites (displayed here in different colors), which are considered to be homogeneous with respect to forest type, aspect and altitude (Price, 2005), image courtesy of Y. Wu.

Applications of VR and IVR are virtual walkthroughs of buildings, virtual prototyping, and entertainment. The *virtual environments* (VE) result from interaction between the cognitive capacity of human and the visual and audible images produced by the computer (Neves et al., 1999). The VEs provided by VR and IVR in particular offer a rich visualization and interaction environment supporting higher levels of abstraction. In scientific applications, hypothesis or models generated with simulation are visualized using visualization; then the results can be analyzed and compared with real data (Van Dam et al., 2000). Biegger (2004) denotes this process as visually supported experimental simulation (cf. Figure 3-7).

The vast majority of VR systems are based on the paradigm of the virtual universe or scene graph. A *scene graph* is a hierarchical arrangement of nodes representing objects, their attributes, and positions in 3-D or 4-D. A node of a scene graph for example groups

objects, objects group several polygons, and polygons group several points or vertices, which is the atomic element carrying the 3 coordinates. The virtual universe originates at an arbitrary centre point of (0,0,0) (Camara et al., 1998; Williams, 1999). The scene graph takes "the hierarchical description of the coordinate systems of each object and builds a tree structure that can be traversed from the root" (Gold et al., 2004) (cf. Figure 3-8). The updated transformation matrix for a new scene is calculated, and sent to the graphics output system, where each object is drawn in turn, after determining the appropriate position. This representation of the hierarchy of coordinate systems is one of the greatest benefits of the scene graph (Gold et al., 2004).

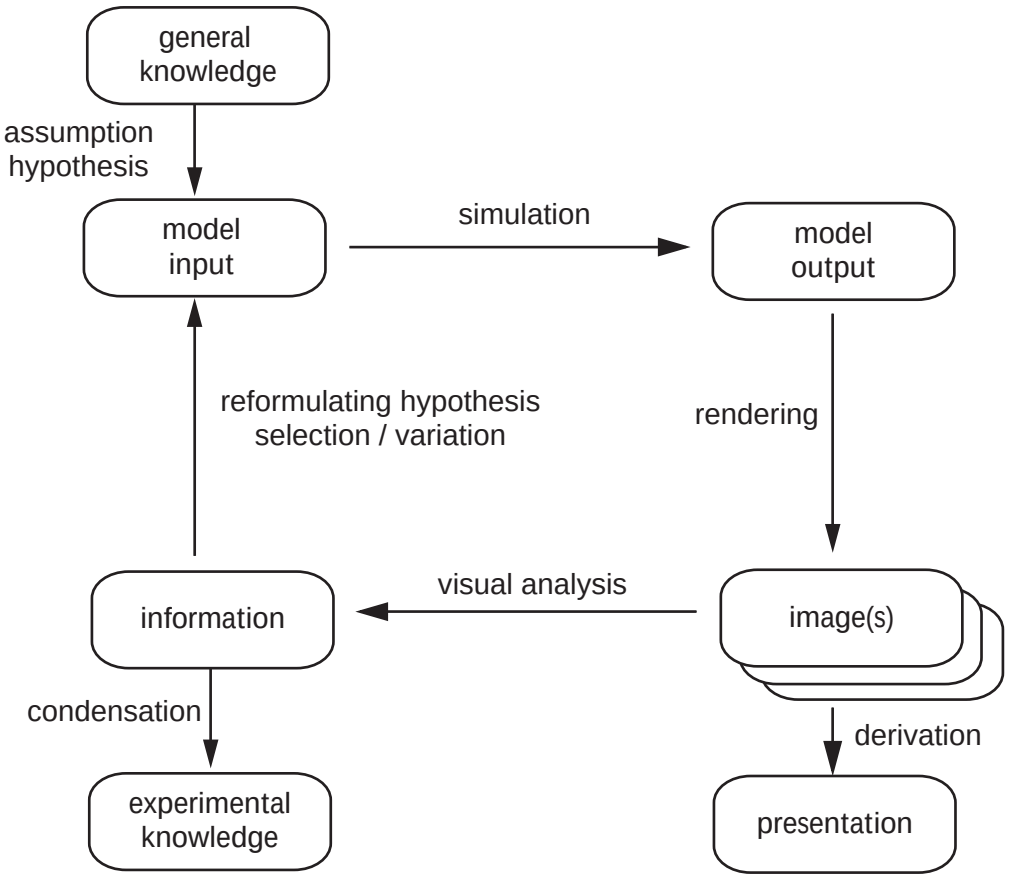

Figure 3-7 The visually supported process of experimental simulation, after Biegger (2004).

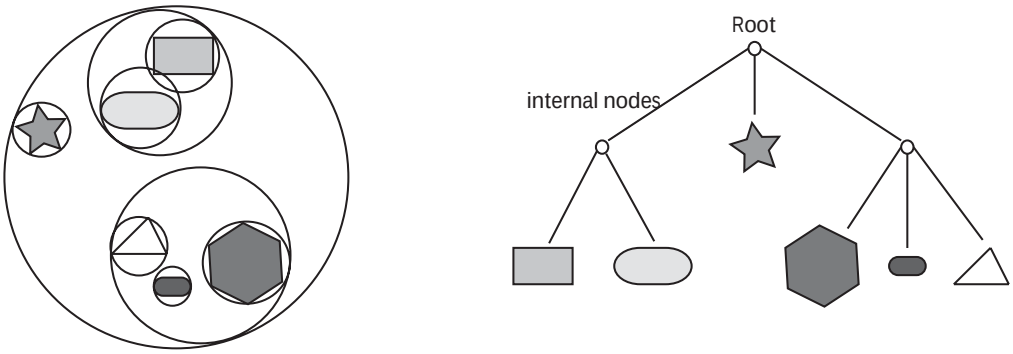

Figure 3-8 The left part shows a simple scene with six objects, each enclosed in bounding sphere, which are grouped together into larger bounding sphere. The right part shows the corresponding scene graph representing the object hierarchy on the left, after Möller and Haines (1999).

The 'data crisis'

The increased data collection capabilities (e.g. image data from remote sensing platforms or spatiotemporal GPS data) led to an exponential rise of availability of data. The information analysis and interpretation efficiency is not advancing to the same extent creating a bottleneck in visualizing data and gaining insights. This is what Van Dam (2000) calls the *data crisis*. Amongst other knowledge discovery techniques, visualization is a powerful tool, because it uses the highly developed human pattern recognition skills. Statistical analysis only shows partial results, and most (standard) GIS graphical user interfaces (GUI) are of the less intuitive windows, icons, mouse, and pop-up menus (WIMP) type (Van Dam et al., 2000). In contrast, VR allows a more holistic understanding of large, complex, and spatiotemporal data. The user can immerse in a virtual world, where navigation and interaction with the data is supported (Kraak et al., 1999). VR and more specifically IVR support real-world experience of the user enabling body-centric judgments about 3-D spatial relations (Van Dam et al., 2000). According to Kraak (1999), Neves (1999), and Van Dam (2000), VR and IVR provide a better environment for the conception, navigation, and exploration of complex 3-D structure.

Scientific visualization and geovisualization

Visualization is suitable for dealing with large, complex, and high-dimensional data using the human pattern recognition skills and the potential of visualization of dealing with a large number of attributes. *Scientific visualization* (SciVis) "transforms numerical or symbolical data and information into geometric computer generated images. It is a methodology for interpreting image data […] as well as data generated from computational models" (Rhyne, 1997). SciVis denotes the process of creating graphical images to increase the human understanding (Kwan and Lee, 2003; McCormick et al., 1987). The focus of research in SciVis is on 3-D computer graphics rendering, time series animation, and interactive, real-time animation (Rhyne, 1997). SciVis is strongly applied in medical imaging, process model visualization, and molecular chemistry (Kraak, 2003).

Geovisualization (visualization of geographic information) is the use of concrete visual representations and human visual abilities to make spatial contexts and problems

more visible (MacEachren and Monmonier, 1992; MacEachren et al., 1999). Geovisualization involves approaches from scientific visualization, exploratory data analysis (EDA), image analysis, and GIS to provide methods and tools for the visual exploration, analysis, synthesis, and presentation (Kraak, 2003). By integrating the geographical dimension in the visualization process, the identification and interpretation of spatial patterns and relationships is facilitated. Most standard GIS concentrate on visualization of 2-D and 2.5-D data, the development of 3-D visualization programs with advanced 3-D modeling and rendering capabilities has been realized mainly outside of the GIS domain (Kwan and Lee, 2003). Geovisualization provides means for presenting geographic data in a virtual environment by rendering *georeferenced objects*, i.e. the objects have georeferenced geometry and semantic information. Usually georeferenced objects are visualized with their spatial reference object, the terrain. In particular, in rugged terrain, as a rule of thumb, 50 percent of the terrain cannot be seen, since the backface of hills is occluded by the frontside. Therefore, geovisualization must be interactive allowing the observer to change his/her viewing point and viewing direction (Biegger, 2004).

In the IPODLAS framework, photorealistic geovisualization is used in the sense of the Robertson's *natural scene paradigm* (Robertson, 1991). The natural scene paradigm involves intuitive understandable models such as 3-D structures or scenes. The natural scene paradigm represents data variables by recognizable properties of the objects or scenes, and tries to enhance the user's cognition by using graphics scene simulation techniques. The usage of this particular model (the natural scene with identifiable physical properties) benefits from the human ability to gain an immediate notion of the model's 3-D structure and the surface coverage (Robertson, 1991). An example of the advantages of natural scene representations is the use of a relief-shaded surface to illustrate the distribution of a scalar variable defined over a 2-D field where the height of the surface represents the value of the variable. A traditional, alternative representation such as a contour map is less intuitive, in particular for a non-expert user (Robertson, 1990).

3.2.2 Interoperability approaches

Most VR developments take place on the basis of *OpenGL*[15] and *Scene Graph*[16]. Besides these rather low-level APIs, no universally accepted interoperability standard exists by which virtual worlds built into different applications can interoperate. As generally in computer graphics, the development of a general standard is hindered by the coexistence of many approaches. The oldest and most established proto-standard for interactive 3-D graphics is the SGI-led OpenGL. Newer packages are *Microsoft's DirectX*[17], *Sun's Java3D*[18], and *W3C X3D*[19] (previously called *VRML*[20]). The dilemma of the interoperability between different approaches is not just the problem of sharing geometries, but also interoperation of behavior and interaction techniques (Van Dam et al., 2000). The application of conversion routines between packages, i.e. different rendering and interaction libraries and visualization toolkits, only produces relief to a certain amount; "conversion routines may

[15] OpenGL (**Open G**raphics **L**ibrary): a cross-language, cross-platform API for 3-D computer graphics (http://www.opengl.org/, accessed December 9, 2005)
[16] cf. http://www.openscenegraph.org/ and http://www.opensg.org/ (accessed December 9, 2005)
[17] http://www.microsoft.com/windows/directx/default.aspx (accessed December 9, 2005)
[18] http://java.sun.com/products/java-media/3D/ (accessed December 9, 2005)
[19] http://www.web3d.org/ (accessed December 9, 2005)
[20] http://www.web3d.org/x3d/specifications/vrml/index.html (accessed December 9, 2005)

be used to link individual packages [...] but this solution becomes inefficient as problem size increases" (Van Dam et al., 2000).

3.2.3 Shortcomings

Despite the gap between the data gathering and the visualization and analysis capabilities, which Van Dam (2000) denotes as 'data crisis', many scientists tend to treat visualization still as a secondary priority compared to investments in computation and data handling. Further, it must be ensured that the links between visualization and the original data persist (Van Dam et al., 2000). For instance, in VTP (cf. subsection 3.2.4) the whole data set representing the terrain must be hold in memory, no database use is supported. Although selecting objects and simple database queries is supported in certain VR applications, there is a lack of more complex analysis functionality (Verbree et al., 1998). The lack of standards and interoperability initiatives in VR hampers the integration of VR applications in software projects.

3.2.4 Legacy system

The tool chosen to act in the virtual reality domain is *Virtual Terrain Project*[21] (VTP). It is an open source project to foster 3-D visualization of real world objects. Besides preprocessing software for geographic data such as terrains, the runtime environment *Enviro* uses scene graph modeling (cf. subsection 3.2.1) and *wxWidgets*[22] for the graphical design of the user interface providing a portable GUI library (cf. Figure 3-9). Cartographic projection is handled by the included *Geospatial Data Abstraction Library*[23] (GDAL) (Biegger, 2004). The goal of VTP is to advance the creation of tools for interactive, 3-D visualization of the real world objects by converging the domains of GIS, visual simulation, surveying, and remote sensing. To support interactive landscape visualization with mobile elements, efficient terrain rendering methods are required. Since VTP satisfies these requirements (Biegger, 2004) and allows as open source project extensions of the existing functionality, it is chosen as VR application in the IPODLAS framework (Wu et al., submitted). VTP can be accessed through a C++-based API and is run in this project in a Windows XP environment.

Another geovisualization system is the commercial LandEx[24]. It is built on OpenGL as low-level rendering layer and offers dynamic and interactive 3-D maps allowing the representing of georeferenced vector or raster data such as digital terrain models and the embedding of additional 2-D or 3-D objects.

GeoVision (Hirtz et al., 1999) is a landscape visualization system concentrating on interactive photo-realistic visualization of the world. It is built on the OpenGL Performer[25] (formerly IRIS Performer). Different methods for the realistic modeling of landscape elements such as terrain, buildings, and meteorological effects are implemented (Biegger, 2004).

---

[21] http://www.vterrain.org/ (accessed January 11, 2006)

[22] WxWidgets (formerly known as WxWindows) is an open source, cross-platform widget library for building graphical user interfaces (http://wxwidgets.org/, accessed January 11, 2006)

[23] GDAL is a translator library for raster geospatial data formats. As library, it provides a single abstract data model for all supported formats. The related OGR library provides similar capability for vectors (http://www.remotesensing.org/gdal/, accessed January 11, 2006).

[24] http://www.hpi.uni-potsdam.de/vrs/landex/ (accessed September 20, 2006)

[25] http://www.sgi.com/products/software/performer/ (accessed September 20, 2006)

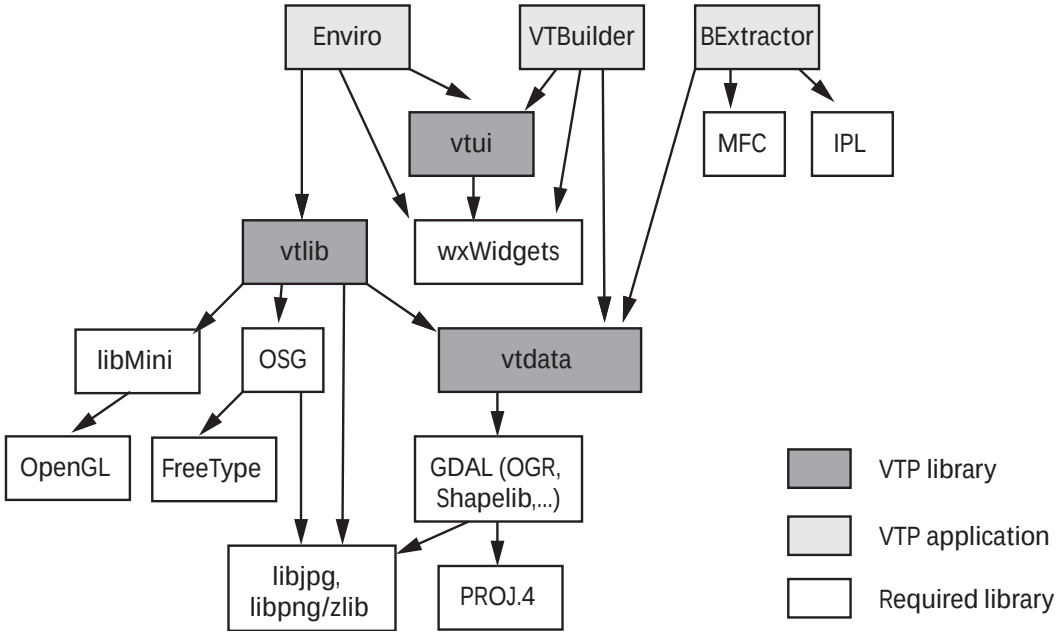

Figure 3-9 The architecture of VTP[26]. The applications Enviro, VTBuilder, and BExtractor use VTP libraries such as wxWidgets and vtlib, and third-party libraries such as GDAL or OpenGL.

3.3 Geographic Information System (GIS)

Longley (Longley et al., 1999) states that "Geographical information is information about geography, that is, information tied to some specific set of location on the Earth's surface". *Spatial* or *geospatial* are terms often used synonymously to geographical. The reference of data to spaces representing the Earth's surface turns data into spatial data and in this sense in two crucial respects into "special" data as Anselin (1989) argued. First, this is the spatial dependence or autocorrelation, which can be expressed using Tobler's 'First Law of Geography' (Tobler, 1970): "everything is related to everything else but near things are more related than distant things". This property of the data violates the principle of independence used in classical statistics. Second, this is the spatial heterogeneity, which means that the characteristics of geographical data change such as that conditions at one place are not the same as conditions elsewhere. This is known as non-stationarity in statistics (Longley et al., 1999).

3.3.1 Basics

The nature of GIS

GIS is perceived as a software environment where geographical information is handled, often in multidisciplinary and integrative use (Longley et al., 1999). GIS provides powerful functionality for collection, analysis, integration, storage, and display of spatial data from different sources (Fedra, 1993; Jones, 1997; Pang and Shi, 2002).

[26] The Figure shows a simplified version of the architectural overview of VTP. The complete version can be found on http://www.vterrain.org/Implementation/index.html (accessed January 17, 2006).

Nyerges (1993) identifies three perspectives under which a GIS can be seen: the functional, the procedural, and the structural perspective. The *functional perspective* addresses the character of GIS use. Onsrud (1989) defines a framework for a taxonomy of GIS usage, which comprises the dimensions:

- Type of task: for example environmental resource management including inventorying, assessing, managing, and predicting
- Application area: environmental, socioeconomic, etc.
- Level of decision: policy, management, operations
- Spatial extent of problem: from small to large
- Type of organization: public, private, etc

Another dimension of the functional perspective of GIS is the mode of GIS usage, where Nyerges (1993) distinguishes mapping, querying, and modeling. In the map mode referential information and browsing of information is provided using standard methods such as pan and zoom. In the query mode, the user queries information about locations and/or phenomena. Finally, in the model mode, GIS support the modeling of processes using tools such as Tomlin's cartographic modeling language (also known as *Map Algebra)* (Tomlin, 1990) exploiting the analytical functionalities of GIS (Albrecht, 1997; Nyerges, 1993).

The *procedural perspective* in Figure 3-10 refers to the nature of the GIS workflow, which consists primarily of four steps: project definition, data input and capture with subsequent data storage and data management, data manipulation and analysis, and data output and display (Nyerges, 1993). The data storage and management functionality are used by all other steps.

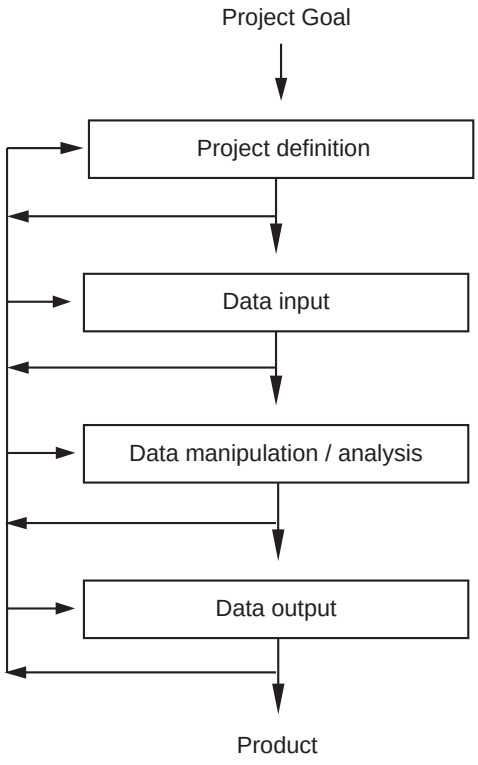

Figure 3-10 Procedural perspective: GIS as a work-flow process (Nyerges, 1993).

A taxonomy of analytical GIS functionalities is given by Albrecht (1997). Analyzing user interfaces (UI) of GIS, he distinguishes 20 universal analytical GIS functionalities categorized into six groups (cf. Table 3-3): search, location analysis, terrain analysis, distribution / neighborhood, spatial analysis, and measurement (Albrecht, 1997).

GIS functionality group	Analytical functionality
Search	Interpolation Thematic search Spatial search (Re-)classification
Location analysis	Buffer Corridor Overlay Thiessen / Voronoi
Terrain analysis	Slope / aspect Catchment / basins Drainage / network Viewshed analysis
Distribution / neighborhood	Cost / diffusion / spread Proximity Nearest neighbor
Spatial analysis	Multivariate analysis Pattern / dispersion Centrality / connectedness Shape
Measurements	Measurements

Table 3-3 The twenty universal, analytical GIS functionalities, after Albrecht (1997).

The *structural perspective* examines the nature of the software architecture of GIS, which can be seen at an abstract level as a composition of generic subsystems (cf. Figure 3-11). The subsystems are a reflection of the processing activities in a GIS: human interface, input/capture, manipulation/analysis, output/display, and data management. The data model used in the data management subsystem determines the constructs for storage, the operations for manipulation, and the integrity constraints ensuring the validity of data stored. The data management subsystem is linked directly to the other subsystems. Its architecture, influenced by the data model, is crucial for the data representation presented to the subsystems or to other systems linked to GIS (Nyerges, 1993).

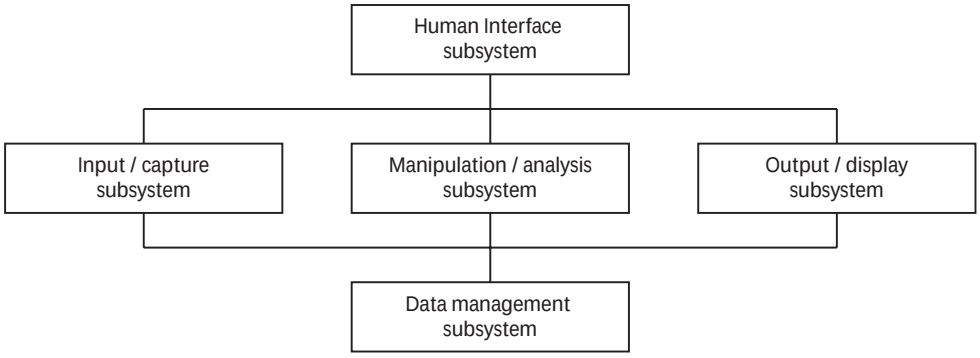

Figure 3-11 GIS software architecture as a composition of subsystems seen from the structural perspective (Nyerges, 1993).

Models of spatial information

The different conceptual models of space emphasize different aspects of the spatial phenomena depending on the purpose of the usage. Jones (1997) distinguishes three distinct models of geographical information (cf. Figure 3-12):

- Object-based models: the emphasis is on discrete objects or entities; the space is perceived as populated by potentially overlapping geometric objects having attributes characterizing them. Individual object are viewed in isolation or in terms of their relationships with other objects; an object can be composed from other objects and can be separated conceptually from other objects. Object-based models are appropriate when considering phenomena with well-defined and boundaries, for example land-use or land-cover modeling.
- Network-based models: also this model often deals with discrete objects, but the focus is less on the shape of the objects, but on interactions between multiple objects taking place on discete paths that connect the individual objects. Typical applications are hydrological and traffic models.
- Field model: for each location a value is assigned by one or more spatially continuous functions. This model is appropriate for phenomena that are considered as continuously variable across space and which do not exhibit crisp boundaries. Examples for the application of a field model is appropriate are the modeling of the temperature or the concentration of pollutants in the atmosphere.

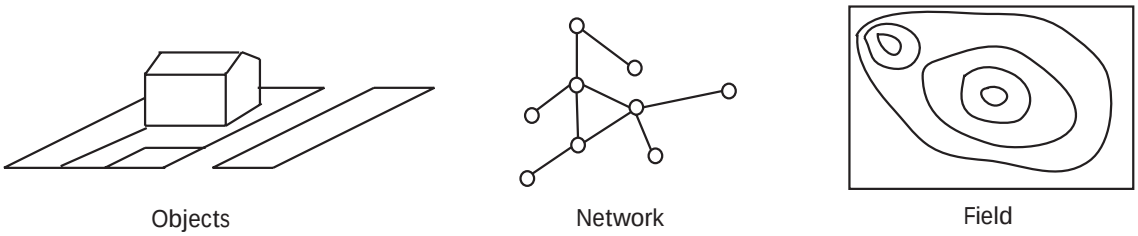

Figure 3-12 Three conceptual models of spatial information, after Jones (1997).

GIS and Time

GIS are primarily designed to model static representations. The data model is centered on representations of the geographical space, the objects located there, and their relationships to each other. The focus is on location and topology (Fedra, 1996). Couclelis (1999) identified the seamless integration of space and time and the accommodation of multiple spatiotemporal perspectives as key challenges in the domain of spatiotemporal representation. Two basic types of questions posed to a model integrating time and space are the ones concerning the world state and change (Peuquet, 1999):

1. World state: What is the spatial distribution of a phenomenon at a given time?
2. Change: Which elements have changed during a given time span?

Peuquet (1999) distinguishes three basic concepts for representing spatiotemporal data in GIS: location-based, entity-based, and time-based representation. *Location-based* representation of spatiotemporal data is the most common (and often the only) one in existing commercial GIS. This form of spatiotemporal representation often employs a grid data model. A series of *snapshots* record the whole state of the considered area for several points in time (cf. Figure 3-13).

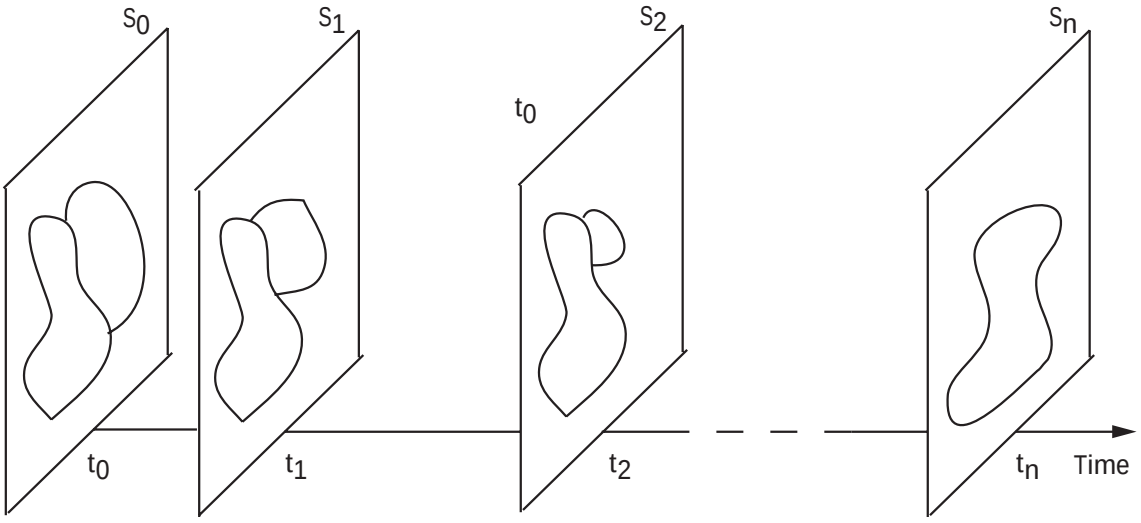

Figure 3-13 Snapshot approach: each snapshot S_i represents the state for a given point in time t_i , after Peuquet and Duan (1995).

The approach is straightforward, the state of any location or entity can easily be retrieved for the recorded points in time. A drawback is that a significant amount of the stored data is redundant, since in each snapshot the whole area (not only the changes) is stored. Furthermore, changes are only recorded implicitly and have to be determined in a costly snapshot-to-snapshot comparison. Changes with smaller temporal resolution as the snapshot frequency are not registered properly. Moreover, the exact time of change cannot be retrieved from the snapshots (Peuquet, 1999).

A modification of the snapshot model just records an initial snapshot and then only changes related to the specific location avoiding the storage of redundant information. An example is the *Space-time composite* model shown in Figure 3-14 (Langran, 1992).

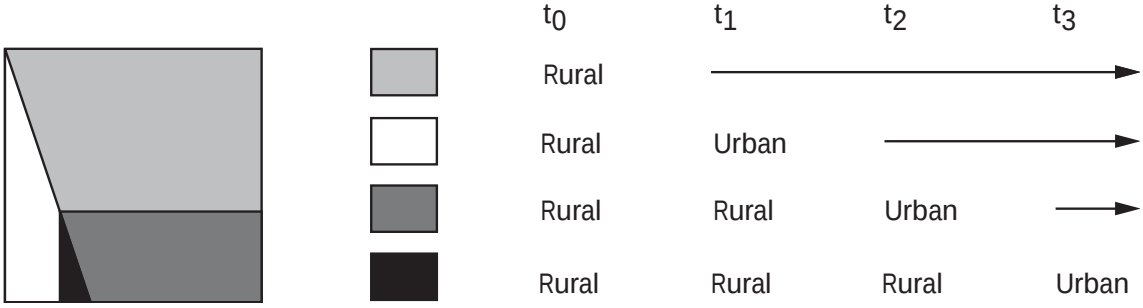

Figure 3-14 Space-time composite of an urban encroachment, each polygon has a different attribute history. For example, in the white polygon the land coverage was at time t_0 rural, then at time t_1 it changed to urban and remained urban till t_3, after Langran (1992).

The *entity-based* representation of spatiotemporal data is conceptually an extension of the object-based model approach. Instead of recording changes of locations, spatial changes of specific geographic entities are explicitly recorded trough time. The spatiotemporal models apply the concept of amendments, changes to the entity are recorded in what Langran (1992) describes as *amendment vector* (cf. Figure 3-15). This approach maintains the integrity and the changing topologies of the individual entities over time. It can also represent asynchronous changes. Drawbacks are the quickly increasing complexity of the space-time

topology and the problems that occur when entities split or merge over time. For these problems, applying the concepts of encapsulation and inheritance of the object-oriented approach (cf. 2.2.2) can produce relief to a certain extent. Object-orientation supports the joint description and handling of spatial, temporal, and attribute properties and allow the representation of complex spatiotemporal relations (Peuquet, 1999; Wytzisk, 2003).

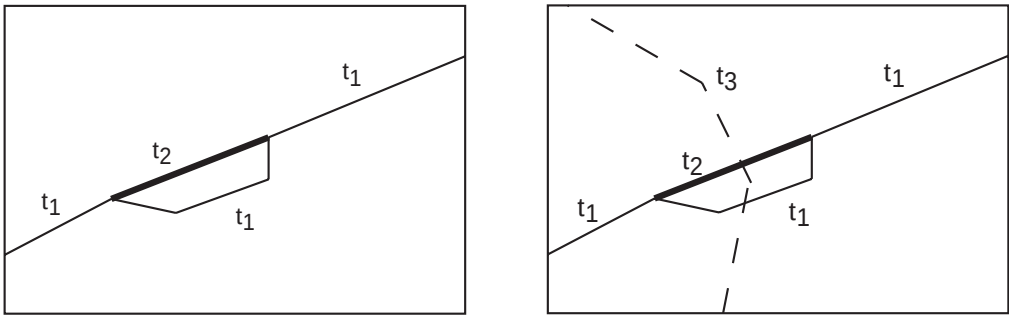

Figure 3-15 The amendment vector approach shows the historical sequence of a road. The thin line of time t1 shows the original route of the road. At time t2 the road was straightened requiring cutting the original line segment and inserting a new one. At time t_3 a new road was built inserting intersection points, after Peuquet (1999).

In the *time-based* representation of spatiotemporal data, time is used as the organizational basis (Peuquet and Duan, 1995). Changes are stored as a sequence of events through time in a temporal vector, each item of the temporal vector has an associated set of locations and entities that have changed at that particular time. Figure 3-16 shows an example for the *event-based spatiotemporal data model* (ESTDM) (Peuquet and Duan, 1995), which defines an event chain describing the spatiotemporal characteristics of the phenomenon of interest. Besides sudden changes that can be recorded in a straightforward manner, for gradual changes a virtual change event has to be defined when the amount of accumulated change has reached a defined threshold. A major advantage of this approach is its support for time-based queries, e.g. retrieve all events that occured between January 1 and June 31 (Peuquet, 1999).

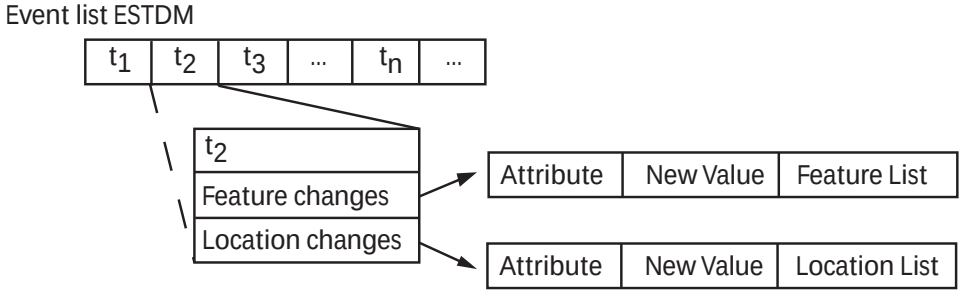

Figure 3-16 Event list ESTDM: changes are stored as sequence of events trough time, after Peuquet (1999).

Different types of representation of spatiotemporal data provide different views and support different types of queries. For example, associating temporal information with individual entities facilitates querying entity histories, while associating temporal information with locations supports working with the history of locations (Peuquet, 1999).

GIS supporting the handling of spatiotemporal data are considered as *Temporal GIS*. Wytzisk (2003) classifies GIS depending on the number of supported spatial and temporal dimensions:

- 2-D GIS support two spatial dimensions. This is up to now the most current category of (commercial) GIS.
- 3-D GIS allow representing 3-D coordinates. A special case of 3-D GIS are the 2.5-D GIS, which represent the third dimension as additional attribute.
- 2+1-D, 2.5+1-D, 3+1-D, respectively 4-D GIS add the temporal dimension to the corresponding GIS.

3.3.2 Interoperability approaches

To reach interoperability on the level of spatial data, special *data definition languages* (DDL) have been established addressing the problem of static and inflexible data formats. They provide an enhanced flexibility in terms of expressivity of the data exchange formats. The data consumer needs to understand the data definition in order to be able to use the data. An example of a DDL for spatial data is *INTERLIS*[27], a data exchange mechanism based on a data model allowing a specification of a data format, i.e. an encoding of the data model (Vckovski, 1998). The main drivers for the specification of interoperable GIS are currently the *Technical Committee 211 of the International Organization for Standardization*[28] (ISO/TC 211) and the *Open Geospatial Consortium*[29] (OGC). To avoid competing or contradictory standards, much work is done in harmonizing their GI models (Gronmo et al., 2000; Wytzisk, 2003). ISO/TC 211 is taking a top-down perspective, using a *Unified Modeling Language*[30] (UML) model-based approach, while OGC employs a bottom-up approach allowing multiple implementation specifications.

ISO/TC 211

ISO/TC 211 specifies a series of standards for geographic information and geomatics. The base series of standards is designed to be independent of any specific implementation environment or distributed computing models and to support data and service interoperability. The implementation-neutral models reside on a conceptual level, but must be precise enough to be mappable to implementation-specific models. The implementation-neutral models are specified using UML according to *ISO 19103 Conceptual Schema Language*[31]. Figure 3-17 shows the ISO/TC 211 approach of getting from the implementation-neutral models to XML-based encoding and service implementations for

[27] INTERLIS (cf. http://www.interlis.ch/index_d.htm, (accessed December 9, 2005)) is e.g. used in the Swiss Official Survey.
[28] http://www.isotc211.org/ (accessed December 22, 2005)
[29] http://www.opengeospatial.org/ (accessed December 22, 2005)
[30] http://www.uml.org/ (accessed December 9, 2005)
[31] http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=37800 (accessed December 9, 2005)

diverse environments, applying *ISO 19118*[32] and *ISO 19119*[33] encoding (Gronmo et al., 2000). [34]

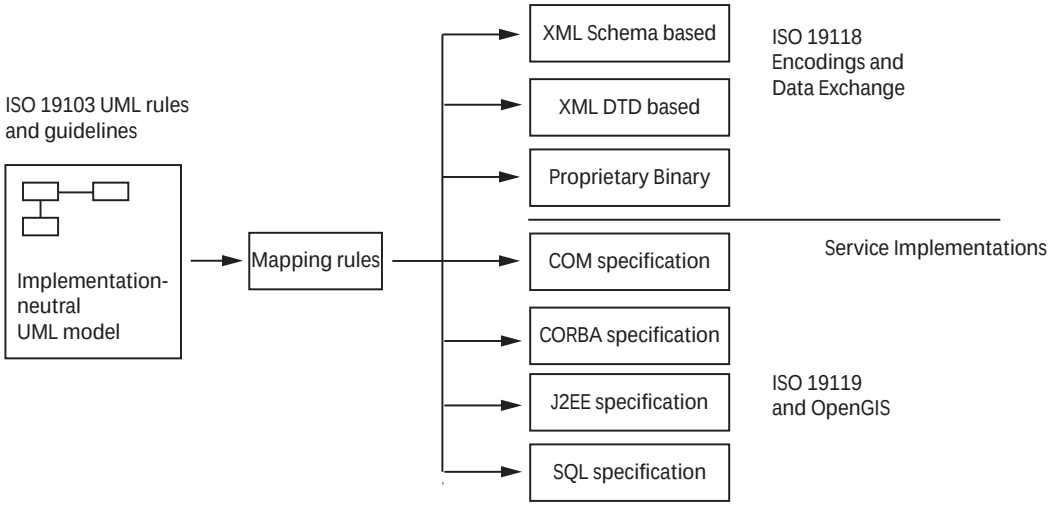

Figure 3-17 ISO/TC 211 models: From implementation-neutral UML model to implementation-specific models, after Gronmo et al. (2000).

In the *ISO 19119 service standard specification* a framework for the specification of geospatial services is defined. In addition, the mappings from implementation-neutral service models to implementation-specific service models for different environments and different distributed component models are specified. Figure 3-18 shows a three-tier architecture that enhances interoperability among different servers and clients by providing an intermediate layer which decouples the communication between clients and servers. The application layer involves browsers and editors offered to the user. The data layer provides representations of the models according to standardized storage representations, such as SQL profiles. The intermediate business layer supports the interaction, that is the communication, between heterogeneous clients and heterogeneous servers. Communication between a COM+[35], a CORBA, and a J2EE environment (cf. subsection 2.3.2) within the business layer as well within the database environment, such as SQL, is the topic of standardization of the OGC, e.g. in the OGC Simple Feature Specification[36] (Gronmo et al., 2000).

[32] http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=37796 (accessed December 9, 2005)
[33] http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=39890 (accessed December 9, 2005)
[34] XML and distributed component models are explained in subsection 2.3.1 respectively in subsection 2.3.2.
[35] COM+ is an extension to COM.
[36] Simple Features are an Open GIS standard specifying the representation of geographical data, which are based on 2-D geometry with no self-intersections, i.e. points, lines, and polygons. http://portal.opengeospatial.org/modules/admin/license_agreement.php?suppressHeaders=0&access_license_id=3&target=http://portal.opengeospatial.org/files/index.php?artifact_id=13227 (accessed May 2, 2006).

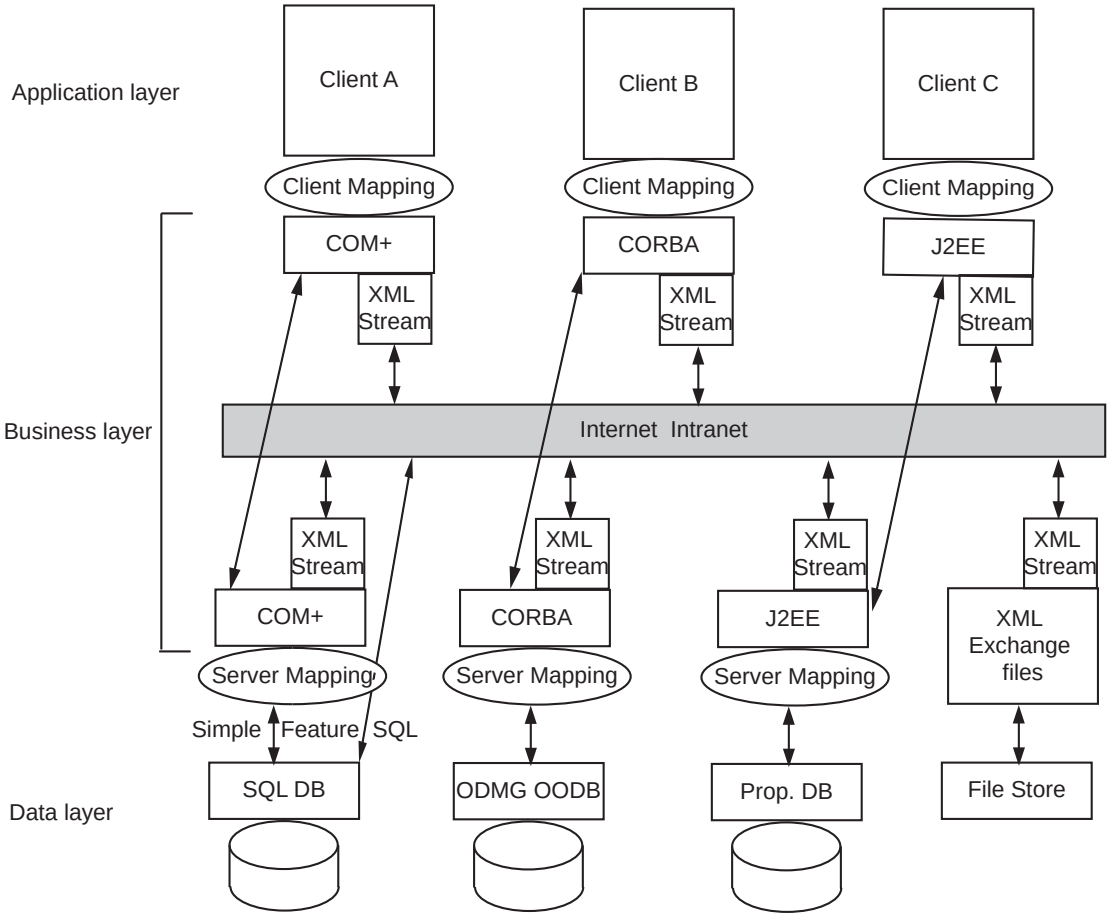

Figure 3-18 ISO/TC 211 service architecture, after Gronmo et al. (2000).

Open Geospatial Consortium (OGC)

The Open Geospatial Consortium[37] (OGC) is an international standards organization developing and implementing standards for geospatial content and services, GIS data processing and exchange. OGC defines platform independent, generic interfaces making up a framework supporting interoperability for GIS components (Bernard, 2001; Buehler and McKee, 1998). The OGC specifications establish a framework defining standardized interfaces for interoperable software components for the handling of spatial information. The platform independent *Open Abstract Specifications* are the basis for the derivation of the distributed component model-specific *Implementation Specifications*. The OGC specifications consist of three parts (Bernard, 2001; Buehler and McKee, 1998):

- The *Open Geodata Model* describes general classes for modeling spatial objects in the sense of a universal geodata model (cf. Figure 3-19).
- On top of the specifications of the Open Geodata Model, access and processing services are specified within the *OpenGIS Service Architecture*, which enables the use and exchange of OpenGIS services through the Internet.
- The *OpenGIS Information Community Model* focuses on a common catalog and documentation of spatial objects within their application domain.

[37] OGC was previously called Open GIS Consortium.

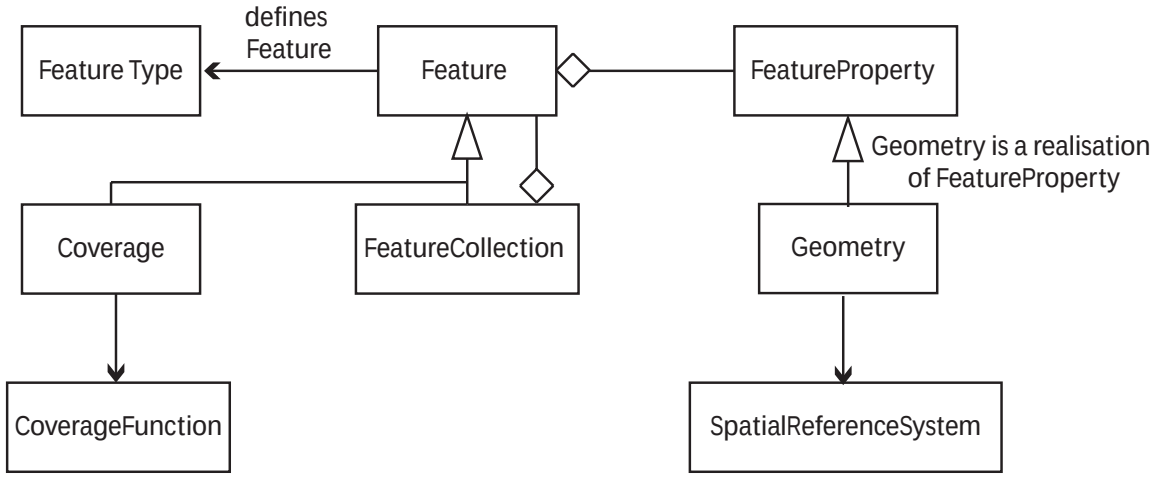

Figure 3-19 Elementary classes of the Open Geodata Model in UML notation, after Bernard (2001).

The Open Geodata model defines an object-oriented, universal spatial data model exhibiting class definitions for the modeling of fields and discrete entities. The top-most base class is *`Feature`*. *`FeatureType`* defines attributes of `Feature`, e.g. types and number of attributes. Attributes are aggregated as instances of *`FeatureProperty`* in `Feature`, which is also the case for the basic spatial information provided in *`Geometry`*. *`FeatureCollection`* is a collection of `Feature` objects and is derived from `Feature`. This means that `FeatureCollection` can contain `Features` of the same type, which in classic GIS is used to represent a thematic layer. However, `FeatureCollection` can also contain `Features` of different types or other `FeatureCollections`. This property is used to model hierarchies. *`Coverage`* associated with a defining *`CoverageFunction`* is used to represent fields (Bernard, 2001).

The Reference Model of Open Distributed Processing[38] (RM-ODP) constitute the basis for the OpenGIS Service Architecture (Percivall, 2002). The architecture defines specifications for geoinformation services which support the integration of services into standard applications (Bernard, 2001). A service is defined as a set of interfaces in a platform independent way. From there distributed component model-specific implementation specifications are derived. Wytzisk (2003) distinguishes in a geographic service taxonomy five types of *geoinformation services*:

- Geographic human interaction services constitute user interfaces (UI).
- Geographic model/information management services allow access and manipulation of geodata and associated metadata.
- Geographic user processing services can be used to provide user-specific services. Geographic workflow/task services support the user to build service

[38] ISO/IEC 10746 Open Distributed Processing Reference Model, ISO, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20696.

chains from different geographic services. Service chains are composed of several interoperating services.

- Geographic processing services modify geodata. Shared processing services provide general functionality; user processing services are user specific.
- Geographic communication services are used to exchange geodata between geographic services.

OGC specifies a logical 4-tier architecture to support the flexible application of geoinformation services. Figure 3-20 shows the relations between the logical, 4-tier OpenGIS Service Architecture and the corresponding physical 2-tier and 3-tier architectures[39] (cf. subsection 2.2.1).

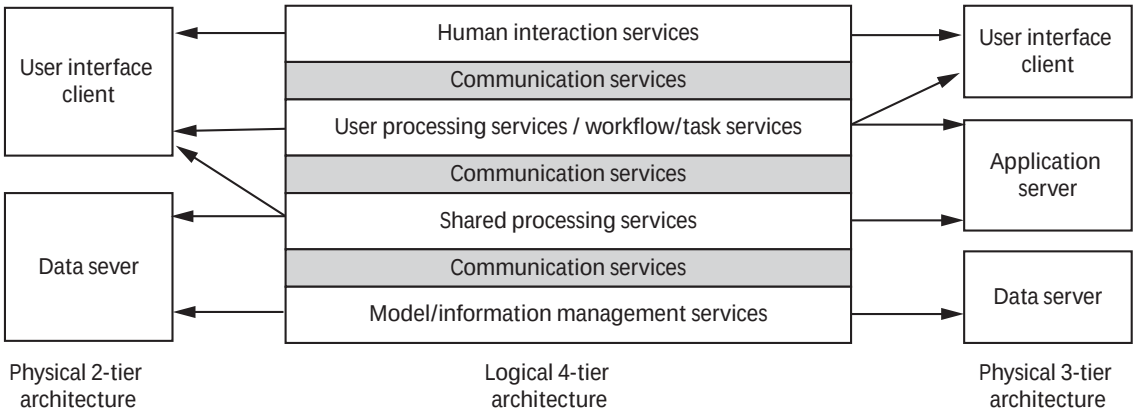

Figure 3-20 The logical, 4-tier OpenGIS Service Architecture and the corresponding physical 2-tier and 3-tier architectures, after Wytzisk (2003).

Applying the web service approach (cf. subsection 2.3.2) the OGC proposes several specifications and implementations of geographic web services. Applying the *GetCapabilities interface* the services provide metadata which decribes the usage of the service. Geodata can be exchanged using GML (Chang, 2006). Well-known geographic web services are:

- *Web Map Service*[40] (WMS): generates maps of spatially referenced data dynamically from geographic information suitable to display on a screen, e.g. pictorial formats such as GIFs or JPEGs or vector-based formats such as SVG.
- *Web Coverage Service*[41] (WCS): extends the WMS interface to allow access to geospatial raster data. Unlike to WMS, WCS provides available data together with their descriptions and allows complex queries.
- Web Feature Service[42] (WFS): allows requests for and manipulation of geographical features encoded in GML. Via this interface, geodata from different sources can be retrieved, combined, and managed.

Geography Markup Language (GML)

OGC developped the *Simple Feature Specification* based on the feature and geometry models of the Open Abstract Specification for sharing geospatial information and providing

[39] Logical architectures focus on composition and interfaces, physical architectures on deployment on hardware components.
[40] http://www.opengeospatial.org/standards/wms (accessed September 21, 2006)
[41] http://www.opengeospatial.org/standards/wcs (accessed September 21, 2006)
[42] http://www.opengeospatial.org/standards/wfs (accessed September 21, 2006)

geospatial services (Preston et al., 2003). The *Geography Markup Language*[43] (GML) (Lake et al., 2004) is an XML extension for encoding the modeling, transport and storage of geographic features, including both the spatial and the nonspatial properties. The key concepts used by GML to model the world are drawn from the OpenGIS Abstract Specification and the ISO 19100 series. GML provides a variety of kinds of objects for describing geography including features, coordinate reference systems, geometry, topology, units of measure and generalized values (Neumann and Eckstein, 2002; Preston et al., 2003). As Figure 3-21 illustrates, the GML core schemas provide a framework for describing geographic objects; this framework is used to define specific geographic objects in the application schemas, which are essentially "vocabularies" (Lake et al., 2004). GML is expected to lead to greater interoperability, e.g. by sharing geographical information within the GIS world (Preston et al., 2003), since GML is based on a common model of geography (i.e. the Simple Features Specification). Being a member of the XML family, GML provides schema validation to verify data integrity (Hoheisel, 2002). With the advent of GML 3 (Lake et al., 2004) the use of temporal information and dynamic features is supported, i.e. there are structures to store and transport temporal information (Preston et al., 2003). GML 3 provides specifications for dynamic features and primitives for representing temporal instants and periods (Lake et al., 2004).

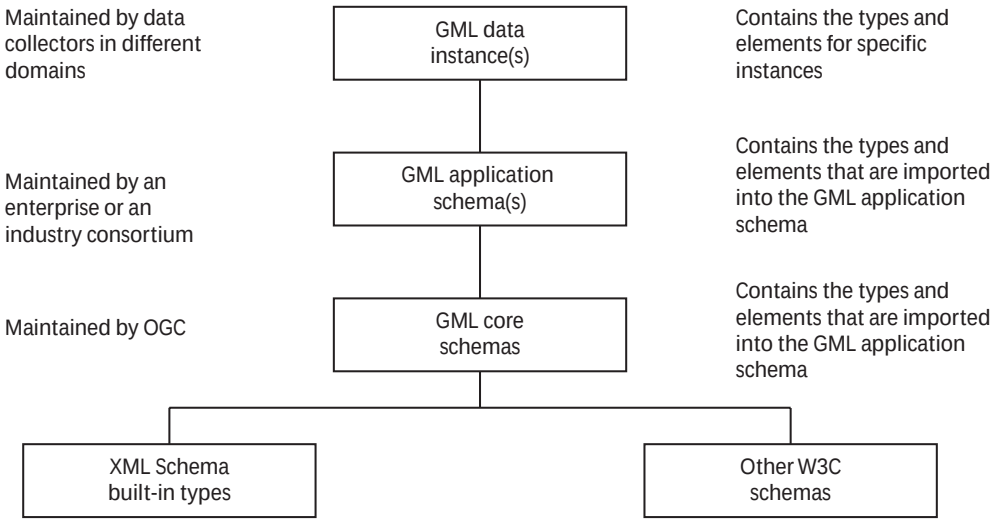

Figure 3-21 A network consisting of XML Schemas, GML core and application schemas, and GML data, after Lake et al. (2004). Each schema imports types and elements from underlying schemas.

3.3.3 Shortcomings

The concepts of landscape representation of classic GIS are predominantly *static*; they represent the space as only existing in the present. Information can be added or changed, but change or dynamics cannot be modeled explicitly. In general, spatially oriented applications cannot effectively model dynamic spatial processes. These applications lack representing dynamics (De Vasconcelos et al., 2002; Pang and Shi, 2002; Peuquet, 1999; Peuquet and Duan, 1995), very much influenced by the cartographic concepts using series of snapshots to record states of spatial aspects (Chen and Jiang, 2000; Peuquet, 2001). For

[43] http://opengis.net/gml/ (accessed January 11, 2006)

example in Spatial Decision Support Systems (SDSS) (Bennett, 1997; Xuan et al., 1998), where complex environmental relations are investigated, the analysis of cause-and-effect relationships is crucial. This ability strongly depends on the analysis of change over time and of patterns of change over time (Peuquet, 1999). On the practical level, a problem users often have to cope with, is how to relate information to past events or how to keep geographical databases up-to-date without overwriting outdated information (Peuquet, 1999). However, standard GIS provide currently some temporal functionality such as ArcGIS[44] with managing histories using databases through ArcSDE[45] or GRASS (cf. subsection 3.3.4) applying timestamping. The missing holistic representations including spatiotemporal aspects as well as the missing synchronization capabilities with distributed spatiotemporal simulation models have hampered the emergence of common integration standards of GIS and TSS (Schulze et al., 2002).

Traditionally, GIS are designed as *closed, isolated applications* that are not compatible with each other. Their reuse for new applications is difficult due to different semantics of data, diversity of data sets, and the heterogeneity of data modeling concepts, data encoding techniques, storage structures, access, and exchange functionalities (Bergmann et al., 2000). Moreover, GIS applications often are very large systems tending to be monolithic and therefore costly to combine with other systems (Preston et al., 2003; Vckovski, 1998). Another impediment are proprietary interfaces. Thus, the benefits of general interoperability standards based on standardized, interoperable, and reusable components cannot be exploited in full extent (Schulze et al., 2002).

The OGC and ISO/TC 211 specifications suffer from the same drawbacks as the GIS in general. Geoinformation is seen in a static way; the interoperability standards only rudimentary define the representation of temporal aspects in an explicit way. Specifications for simulation and analysis methods for spatiotemporal phenomena are lacking or not sufficient (Bernard, 2001; Schulze et al., 2002; Wytzisk, 2003).

3.3.4 Legacy system

GRASS[46] (Geographic Resources Analysis Support System) is an open source GIS with raster, topological vector, image processing, and graphics production functionality for 2-D data operating on various platforms. In some modules also 3-D data and simulation functionalities are implemented. GRASS contains both a graphical user interface (named *TCLTKGRASS*) and command line oriented user interface (Neteler and Mitasova, 2002). Like many commercial GIS, GRASS GIS provides broad data integration functionality and supports a wide range of raster and vector formats including OGC compliant formats to increase its interoperability capabilities (Neteler and Mitasova, 2002). As Figure 3-22 shows, in the GRASS 5.1 vector architecture attributes can be managed in SQL-based databases (Neteler and Mitasova, 2002), such as *PostgreSQL*[47] or *MySQL*[48] (Blazek et al., 2002). GRASS has been chosen as GIS applied in the IPODLAS framework because its modular architecture supports easy access of its functionality via scripting. Second, GRASS provides

[44] http://www.esri.com/software/arcgis/index.html (accessed May 30, 2006) and http://downloads.esri.com/support/documentation/ao_/698What_is_ArcGIS.pdf (accessed May 30, 2006)
[45] ArcSDE is a part of ArcGIS and acts as the GIS gateway to spatial data stored in an RDBMS (http://www.esri.com/software/arcgis/arcsde/, accessed January 03, 2006).
[46] http://grass.itc.it/ (accessed January 25, 2006)
[47] http://www.postgresql.org/ (accessed January 11, 2006)
[48] http://www.mysql.com/ (accessed January 11, 2006)

a broad range of GIS functionality. In this project, a GRASS 5.7 release managing its non-spatial attributes in a PostgreSQL database (Release 7.3.2) was employed. The GRASS functionality is mainly accessed using the command line interface applying Python or Bash scripts.

Another well-known exponent of GIS is the commercial ArcGIS[49] which is a integrated collection of GIS software tools. The ArcGIS framework consists on the desktop GIS level of several components, e.g. ArcView[50], a spatial data viewer, and ArcInfo[51] which provide capabilities for data manipulation, editing, and analysis. On the server GIS level, ArcGIS provides ArcGIS Server, a server-based GIS allowing access via browser and ArcIMS, which is an internet map server delivering maps over the internet. The current ArcGIS release 9.1 applies the Microsoft COM standards allowing scripting languages such as Python access to GIS processing tools.

Quantum GIS[52] (QGIS) is an open source GIS which concentrates on mapping geographical data. However, numerous plug-ins are available to extend its functionality. QGIS is an efficient multi-platform application requiring less disk space and processing power than comparable GIS.

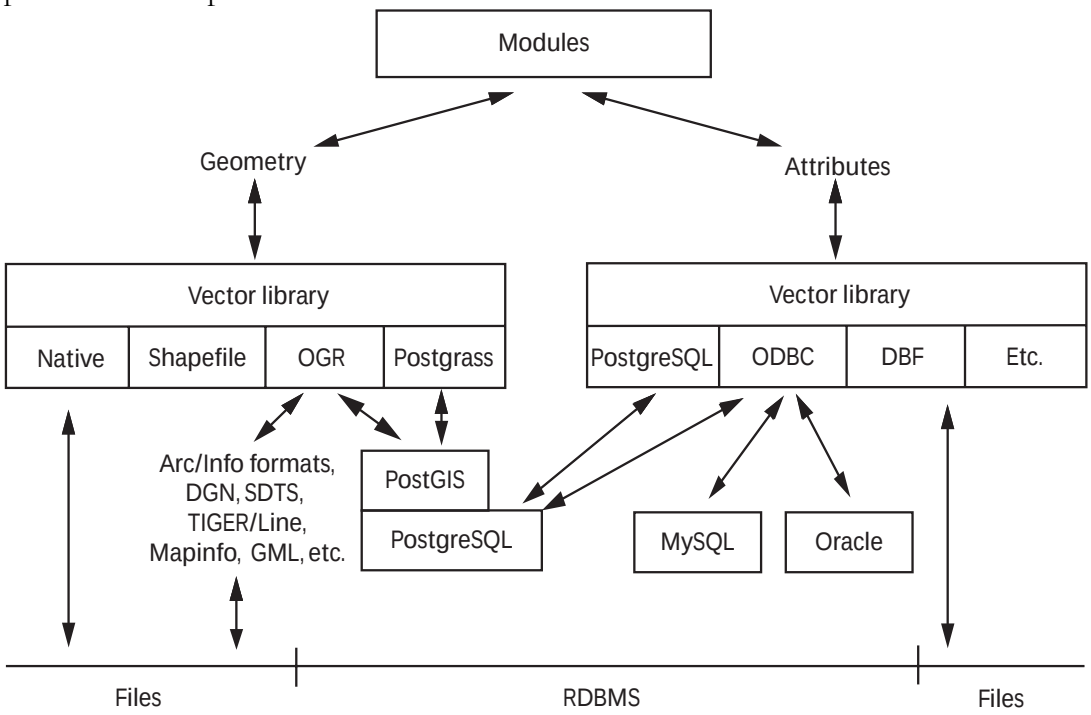

Figure 3-22 Vector architecture of GRASS GIS 5.1 (Blazek et al., 2002) with geometry and attribute library. GRASS can integrate various formats describing the geometry (inter alia ESRI Shapefiles or GML) and attribute information stored in standard databases, such as PostgreSQL or MySQL (Blazek et al., 2002).

[49] http://www.esri.com/software/arcgis/index.html (accessed September 20, 2006)
[50] http://www.esri.com/software/arcgis/arcview/index.html (accessed September 20, 2006)
[51] http://www.esri.com/software/arcgis/arcinfo/index.html (accessed September 20, 2006)
[52] http://www.qgis.org/ (accessed September 20, 2006)

3.4 Combination of TSS, VR, and GIS

The overarching axiom of this thesis is that combined use of functionality of the domains TSS, VR, and GIS to represent and model spatiotemporal, cross-scale processes bears the promise of gaining a more holistic view compared to the rather fragmented view when using the applications of the respective domains in an isolated, sequential mode. TSS models dynamic relationships between cause and effect, VR represents data in a way that facilitate users to identify patterns and relationships, and GIS provides functionality to examine spatial relationships among objects (Wang, 2004). By applying the respective subsystem for the usage it is designed for, the mutually complementary strengths and weaknesses of TSS, VR, and GIS can be exploited in a beneficial way. The hybrid approach promises to apply the best of the three worlds in terms of flexibility and capability. In SDSS and in planning, there is a considerable amount of experience in using TSS, VR, and GIS in research and in practice, but few have met the challenge of integrating all three technologies in one system (Wang, 2004). However, while there is substantial know-how and a long history of combining TSS and GIS (Bernard, 2001; Brimicombe, 2003; Fedra, 1993; Fedra, 1996; Goodchild, 1996; Mladenoff et al., 1996; Raper and Livingstone, 1995; Vckovski, 1998) and of VR and GIS (Huang et al., 2001; Lindstrom et al., 1997; Pajarola and Widmayer, 2001), considerably less work was dedicated to combine TSS and VR (Wang, 2004).

Steyaert (1993) identifies typical application fields for a combined usage of *TSS and GIS* in environmental modeling: these are atmospheric and hydrological models (Bennett, 1997; Bernard, 2001) or eco-system dynamics models (De Vasconcelos et al., 2002; Fischlin, 1982; Mladenoff et al., 1996; Schöning, 1996). Combined use of *VR and GIS* has been reported for urban planning, environmental planning and impact assessment, and archeological modeling (Williams, 1999). In planning and scenario generation, there is some experience in using *TSS and VR* together (Wang, 2004), for example, VR is used to present and communicate results of simulations (Camara et al., 1998). A complementary, but interesting approach of simulation and visualization of spatial issues is shown by Wood (Wood, 2002). He used Java to simulate and visualizing in an object-oriented way the evolution of an ant colony introducing basic GIS functionality. Within the IPODLAS project, work combining TSS, VR, and GIS have been accomplished by Price (Price, 2005; Price et al., in press; Price et al., submitted), Wu et al. (Wu et al., submitted; Wu et al., accepted), and Isenegger et al. (2005).

Status quo

Combined usage of *TSS and GIS* supports analysis of temporal and spatial relationships and changes together (Wang, 2004). For TSS, the principal benefit of being linked to a GIS is gaining the ability of dealing with large volumes of spatially oriented data. Major environmental tasks such as inventory, assessment, management, and prediction in diverse research areas such as atmospheric modeling, land surface-subsurface modeling, and ecological systems modeling can potentially be supported with GIS functionality (Nyerges, 1993). The usage of all primary GIS functionality including data entry/capture, data storage/management, data manipulation/analysis, and data display/output can be beneficial to a TSS. However, GIS is frequently used only as a pre-processor to integrate and prepare spatially distributed input data (e.g. for parameter estimation), and as a post-processor to display and analyze model results (Nyerges, 1993). The use of GIS functionality for data manipulation, analysis, and presentation of model results of urban

growth and land-use simulation models is shown in Landis (2001) Other tasks are assisting in modeling, e.g. in calibrating parameters (Bennett, 1997; Brimicombe, 2003; Fedra, 1993; Fedra, 1996). Engelen et al. (1999) couples simulation functionality of a CA with GIS functionality.

When *VR and GIS* is used together, GIS may profit from 3-D functionality and time series animation of VR (Rhyne, 1997). VR can benefit from the strong analytical capabilities of GIS, and in turn enhance the static representations in GIS as visualization add-on by dynamic and realistic presentation of spatial data to the user. Other benefits are the improved interaction possibilities with spatial data provided by VR (Wang, 2004). Recent advances in software development such as the move of GIS towards component technology and the increased acceptance of VR due to its availability at low cost and its improved performance has led to an increase of combined use of the two technologies. GIS produces data views and VR is applied to visualize them and provide high-level interactivity with the GIS output. Like this, VR is applied to enhance the user's intuitive cognition of the data and its context. Despite of the positive tendencies, it is generally not possible to access from the VR user interface data structures and functionalities of the GIS, for example manipulating object and their relationships (Williams, 1999). An example for a viewing extension of a standard GIS is ArcScene[53] which is the 3-D viewing application of the ArcGIS. NVIZ[54] is an animation and visualization tool for GRASS allowing for 3-D representations and scripted animation. However, both tools do not provide the same functionality as a state-of-the-art VR application.

VR is also used to represent dynamics modeled in TSS in a realistic visual environment. However, the combination of TSS and VR without using a GIS is less common when dealing with problems from the spatial domain (Wang, 2004). TSS and VR is applied for visualization of non-visible properties of the space or of "what-if" scenarios resulting from simulations, e.g. population densities surfaces (Wood et al., 1999) or generating scenarios for water quality management (Camara et al., 1998).

Whereas the combination of two of the three technologies, in particular TSS and GIS and VR and GIS, has reached some level of maturation, the combined usage of the three domains is less common (Wang, 2004), two of the few examples are Wang (2004) and Camara et al. (1998).

Problems of combinations of TSS, VR, and GIS

Burrough (1988) identified important issues which must be considered when *linking land resource assessment models and GIS*. Based on this compilation of concerns, Table 3-4 shows an extended collection of concerns of the domains TSS, VR, and GIS when combining the three systems.

Domain	Concerns of combination
TSS	What are the basic assumptions and methods? At what scale or organizational level is the model designed to work? What kinds of data are needed for control parameters? Under what conditions are certain control parameters more important than others are?

[53] http://www.esri.com/news/arcuser/0103/files/3display.pdf (accessed on January 4, 2006).
[54] NVIZ is a toolbox consisting of several tools for combining and visualizing GRASS raster and vector (http://grass.itc.it/gdp/html_grass53/html/nviz.html, accessed January 4, 2006).

	How are errors propagated through the model?
GIS	Are the right input data available at an appropriate spatial resolution? Do sufficient data exist for creating an appropriate substrate? Are data available for calibrating and validating the model? If data are not available, can surrogates be used, and how should they be transformed? Is information available on data quality and errors? If the results are not good enough, should the GIS suggest alternative data or alternative models to the user?
VR	What is the coordinate system of the data? How to georeference the geographic data? Is 3-D information available? In which form? How can data be exchanged? What are the data exchange formats? How can interaction in VR be communicated to other systems and vice versa? How can functionality of other systems be accessed?

Table 3-4 Concerns for effective linking of TSS, VR, and GIS based on a initial compilation of concerns for linking TSS and GIS by Burrough (1988).

The concerns that Burrough identified for coupling land resource assessment models and GIS indicates that a combined usage of TSS and GIS is more than just a software system integration problem. It is a challenge involving data, software, hardware, and personnel issues and requires effective institutional arrangements and efforts to make progress (Nyerges, 1993). Focusing on the software aspect of a combination of TSS and GIS, one underlying core problem in moving forward towards a better combined usage are the differing data models used in GIS and TSS (Aspinall and Pearson, 2000; Bennett, 1997; Fedra, 1993; Fedra, 1996). In Brimicombe (2003) the differences are identified as "in the case of GIS, the data model focuses on […] representation of geographical space, the objects contained therein and their spatial relations. […] Environmental simulation models […] are predominantly concerned with spatial processes, their states and throughputs of quantities. One is a static representation, the other is concerned with dynamics". Using discrete object representation in GIS implies hard, crisp, non-overlapping boundaries; TSS is rather interested in responses to environmental gradients. These differing emphases result almost necessarily in different conceptual and implementation structures, e.g. in different storages structures (Brimicombe, 2003) and different input/output formats and tools. The same issues as for domains TSS and GIS in general are true for the dominant standards in TSS and GIS, HLA respective the OpenGIS specifications. The overview in Table 3-5 outlining the characteristics of the OpenGIS web services and HLA illustrates that the required properties for establishing interoperable standards within their respective domain can be provided. Yet, overarching standards specifying access and automated combined usage of spatial and temporal dynamic information are currently not existing (Wytzisk, 2003).

Standard / Framework	General characteristics				GIS concerns				TSS concerns			
	Standardization	Distributed	Web service-oriented	Distributed component model independent	Geodata visualization	Geodata access and management	Geodata processing	Visualization	Event-based	Continuous	Agent-based	Temporal synchronization of distributed simulation
OpenGIS web services	y	y	y	y	y	y	y	y				
HLA	y	y		y					y	e	e	y

Table 3-5 Overview over properties of OpenGIS web services and HLA, after Wytzisk (2003), y: yes, e: implemented in extensions.

As most VR systems work with an arbitrary origin of the virtual universe, they do not provide a standard geographic coordinate system and geo-referencing as standard functionality, but in general, coordinates to VR objects can be assigned. A large amount of GIS data is 2-D, while VR operates on 3-D data. Moreover, most of the current GIS data formats are not compatible with the data formats used in VR, and the few exceptions suffer from subtle differences (Williams, 1999). As classical GIS work with 2-D data, they do not provide functionality for 3-D operations, the information is organized in 2-D layers of fixed scale and degree of detail (an exception are 3-D GIS). VR systems are limited to simple interaction with the GIS as e.g. database queries such as select, but cannot access the whole range of GIS functionality (Haklay, 2001; Williams, 1999). In the past spatial data standards have rarely considered the visualization of data and computer graphics rendering libraries evolved independent of spatial data models (Rhyne, 1997). As in VR no dominant standard exists (Van Dam et al., 2000) there is currently not the same amount of work dedicated to bring standards from VR and GIS together as there is between TSS and GIS.

3.4.1 Integration strategies

The meaning of the term integration between the two technologies TSS and GIS is not trivial. The degree of integration varies in practice from project to project (Brimicombe, 2003). Integration strategies can be observed focusing on different aspects which are described in the following. The enumerated integration options are not independent from each other, i.e. certain combinations are more likely to be useful than others. The integration strategies in this subsection and in subsection 3.4.2 are described for the examples of combining TSS and GIS and VR and GIS. However, they remain on a general conceptual level, so that they also apply for other combinations of applications of the involved domains.

Coupling versus integrating

For the 1990s, Brimicombe (2003) identifies a typology of four integration levels with minor variations between the different authors (Fedra, 1993; Karimi and Houston, 1996; Sui and Maggio, 1999), focusing on data management and data integration (cf. Figure 3-23).

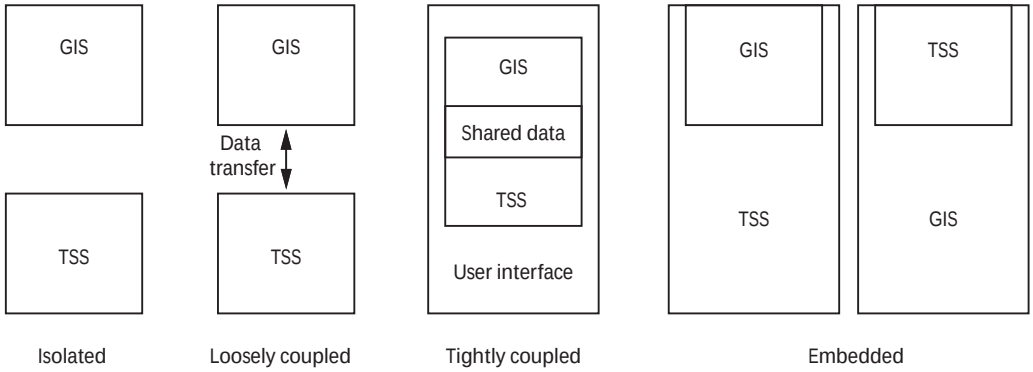

Figure 3-23 Levels of integration between GIS and TSS, after Brimicombe (2003).

Isolated use of GIS and TSS is not really a level of integration, but it describes a situation which is quite common. GIS is used to replace manual measurements, for example distance or area measurements used as a parameter in TSS. Spatially averaged or interpolated values can be produced as input for lumped models. Isolated solutions are characterized by manual data exchange and the use of different data models in GIS and TSS, respectively. In general, GIS is used to generate inputs for TSS (Brimicombe, 2003; Wytzisk, 2003).

In the *loosely coupled* level of integration, data files are shared exporting data in some common export format, for example an industry standard, sometimes even requiring human interaction (Fedra, 1993; Wang, 2004). Each subsystem is running trough its own interface, TSS and GIS do not share a common data model, and the data set as well as the method set is highly redundant. Interaction of the subsystems and between TSS and user is restrained during simulation (Bennett, 1997; Bernard and Krueger, 2000; Wittmann, 2000). This flexible solution is favorable in a unstable software environment where frequent upgrading is likely (Brimicombe, 2003), but due to redundant datasets and functionality sets, the danger of resulting inconsistencies is increased (Bernard and Krueger, 2000). The 'Intelligent Tree' (Isenegger et al., 2005) (cf. Subsection 6.2.2) is an example for loosely coupling exchanging only file but having three individual subsystems with separeted UIs and data sets.

In *tightly coupled* or *deep coupled* systems a common UI allows full access to both functionalities. A common data file exchange format, if not a common data format, enhances seamless data sharing facilitating faster switching from one application to the other. The higher development costs of this approach pay off rather in stable software environment, since each update in one subsystem is likely to cause updates in the other subsystem (Brimicombe, 2003). The final IPODLAS system (cf. chapter 7) exhibits characteristics of a tightly coupled system having a common data exchange format, shared storage, and a common UI providing access to some functionalities of the subsystems. Altough the two applications may appear as one to the users, the applications remain separately (Wang, 2004).

In *embedded* solutions, one subsystem is embedded in the other. The integrated subsystems have one common data model and the data storage is handled centrally (Fedra, 1996). Interaction of the user and the system is possible during simulation and there is only one UI over which the communication with the system is carried out (Bennett, 1997). In partially embedded systems, only the required part of the functionality of one subsystem is implemented in the other. An example for embedding is the use of built-in macro

languages in GIS allowing the implementation of environmental simulation models in GIS. In SPARKS (Schöning, 1996) (cf. Subsection 5.1.2) the Wildland fire simulation model of Rothermel (1972) is implemented using the macro language AML. However, the built-in macro languages are in general significantly slower than the compiled code of stand-alone TSS. A full integration of the functionality of the involved subsystems is costly and therefore relatively rarely aspired to (Brimicombe, 2003; Wittmann, 2000; Wytzisk, 2003). Integrating TSS in a GIS is advantageous, if the structure of the simulation model is not too complex and if the use of GIS functionality is predominant. The opposite, the integration of GIS in TSS, is preferrable, if the GIS functionality is scarcely used during a simulation run (Wittmann, 2000).

Four levels of integration

Rhyne (1997) identifies a classification of combination levels of scientific visualization (SciVis) (cf. 3.2.1) and GIS, which show strong similarities with the preceeding classification of Brimicombe (2003). The four levels of SciVis and GIS integration represent an increasing level of converging the two domains. The *rudimentary* level applies a minimal amount of data sharing and exchange. In the *operational* level consistency of data through removing data redundancies is aspired, this can be achieved by establishing access for SciVis applications to GIS DBs. Transparent communication between the respective software environment is characteristic for the *functional* level. Open GIS data standards and software links may allow SciVis applications to directly access spatial data analysis functionality. The (not yet reached) *merged* level offers comprehensive toolkits, which requires merged underlying concepts of SciVis and GIS.

File-based versus process-based

Focusing on data flow Wittmann (2000) differentiates between subsystems using files for the data exchange and subsystems using processes. As stated in subsection 2.4.1 the implementation of a *file-based* solution is generally simpler than a process-based solution, since the functionality required for establishing file-based data exchange usually is standard. Besides reduced performance, the functionality of the subsystem remains only accessible from the respective subsystem. The data exchange in a loosely coupled solution is likely to be accomplished in a file-based way. *Process-based* data exchange results in better performance and in better accessible functionalities of the involved subsystems. However, the latter solution requires a considerable amount of reimplementation, since the code of the involved functions has to be changed to support processes-based connections with other subsystems. Process-based data exchange is a probable approach for embedded systems.

Common versus independent

Looking on the control flow and the UI, Wittmann (2000) suggests three classes of combinations of TSS and GIS (cf. Figure 3-24). *Independent* controlling of subsystems using their respective UI leaves TSS and GIS unchanged and requires a *transformer* to be implemented. The transformer is controlled by the user on the level of the operating system to convert data from one subsystem to the other. The user must have a profound knowledge of the whole system, since the control flow is entirely in her/his responsibility. Independent UIs is a viable solution for loosely coupling. In the *client-server* approach, the user interacts with the UI of only one subsystem, over which the complete control flow is

handled. This demands the extensibility of the subsystem the user interacts with in order to incorporate user-defined commands and the client-server communication with the other subsystems. This solution can be suited for tightly coupled systems. In the *common user interface* approach, all subsystems communicate with a common UI which controls the interaction of the subsystems. This approach is employed to control the subsystems of the final IPODLAS system (cf. chapter 7).

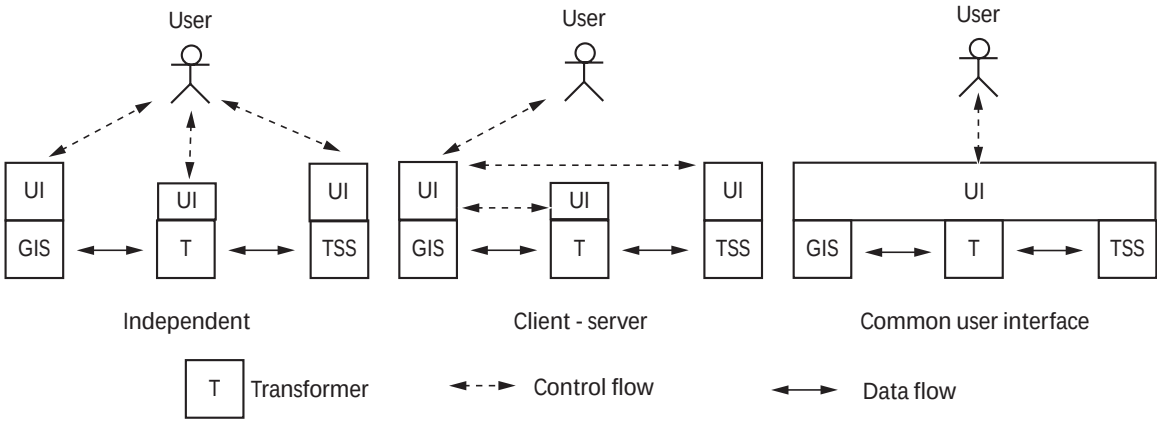

Figure 3-24 Three designs of the control flow and the user interface for combination of TSS and GIS: independent, client-server, and common UI, after Wittmann (2000) .

3.4.2 Integration typologies

Several developments attracting some attention in the recent past influence how the relations between subsystems are conceptualized. One of those developments is the emergence of networks. It changed the view from focusing on (isolated) subsystems to a paradigm where the network is the core technology using functionality of different subsystems like tools from toolboxes (Batty, 1999; Brimicombe, 2003; Coleman, 1999). Another trend is the growing maturity and acceptance of interoperability standards developed in the GIS community by the OGC and ISO/TC 211 and for TSS by HLA.

Integration is seen here as bringing together two rather different technologies into closer proximity trough sharing data and interfaces, so that they can work together (Brimicombe, 2003). In contrast, *interoperability* is the ability of client-side software to use services from server-side applications such that user expectations are fulfilled (Albrecht, 1997). Interoperability allows software components to seamlessly operate together, thus "interoperability is the bringing together of software at a more structural, software developmental level than is usually achieved trough the integration of independently developed software" (Brimicombe, 2003). The mutual interoperability of general software components is supported by the dominance of a small number of operating systems such as Microsoft Windows and several distributions of UNIX and Linux, the emergence of the object-oriented programming paradigm and the use of communication protocols. These factors result first in overarching common principles of UI design. Second, data access is largely transparent (cf. subsection 2.3.2) to the user, since remote data access over a network is facilitated by the widespread application of services for data retrieval, access, extraction, and transformation. Thirdly, through providing the same interface on the major

platforms (e.g. the Java Virtual Machine[55] (JVM)), high-level languages such as Java (Flanagan, 2005) or Python (Martelli, 2003) offer can be used as wrappers for the original software to provide common interfaces (Brimicombe, 2003).

Maturing typology

Considering the developments in technology and the emergence of interoperability approaches, and evolving from different combination approaches for TSS and GIS, Brandmeyer and Karimi (2000) propose a hierarchical typology of integration of TSS and GIS. Brimicombe (2003) generalizes this typology to what he calls a '*maturing typology*': It considers two distinct general subsystems (instead of only TSS and GIS) operating within a specialized software environment, i.e. the two subsystems cannot be assumed to share a common data model and/or common data handling (cf. Figure 3-25). While the typologies discussed above are focused on integrating TSS and GIS, this 'maturing' typology can be considered to be of more general nature. '*One-way data transfer'* is the lowest level of integration. The subsystems are used in a separated mode; a common language, data storage format, or operating system cannot be presumed. The low-cost solution to combine the two subsystems is one-way data transfer. '*Loose coupling'* is similar to the loosely coupled integration level of the typology described in Figure 3-25 providing a usually automated two-way data exchange. At the '*shared coupling'* integration level, one major software component is shared. If the data storage component is shared (*data coupling*), the data storage and the data are shared requiring that both subsystems use the same data model. If the UI is shared (*interface coupling*), the user interacts with a common UI, the internal coupling mechanisms are hidden from him/her.

[55] JVM is available on the major hardware and software platforms acting as middleware decoupling the actual java program from the underlying platform (http://java.sun.com/docs/books/vmspec/ and http://www.javaworld.com/javaworld/jw-06-1996/jw-06-vm.html, accessed January 18, 2006)

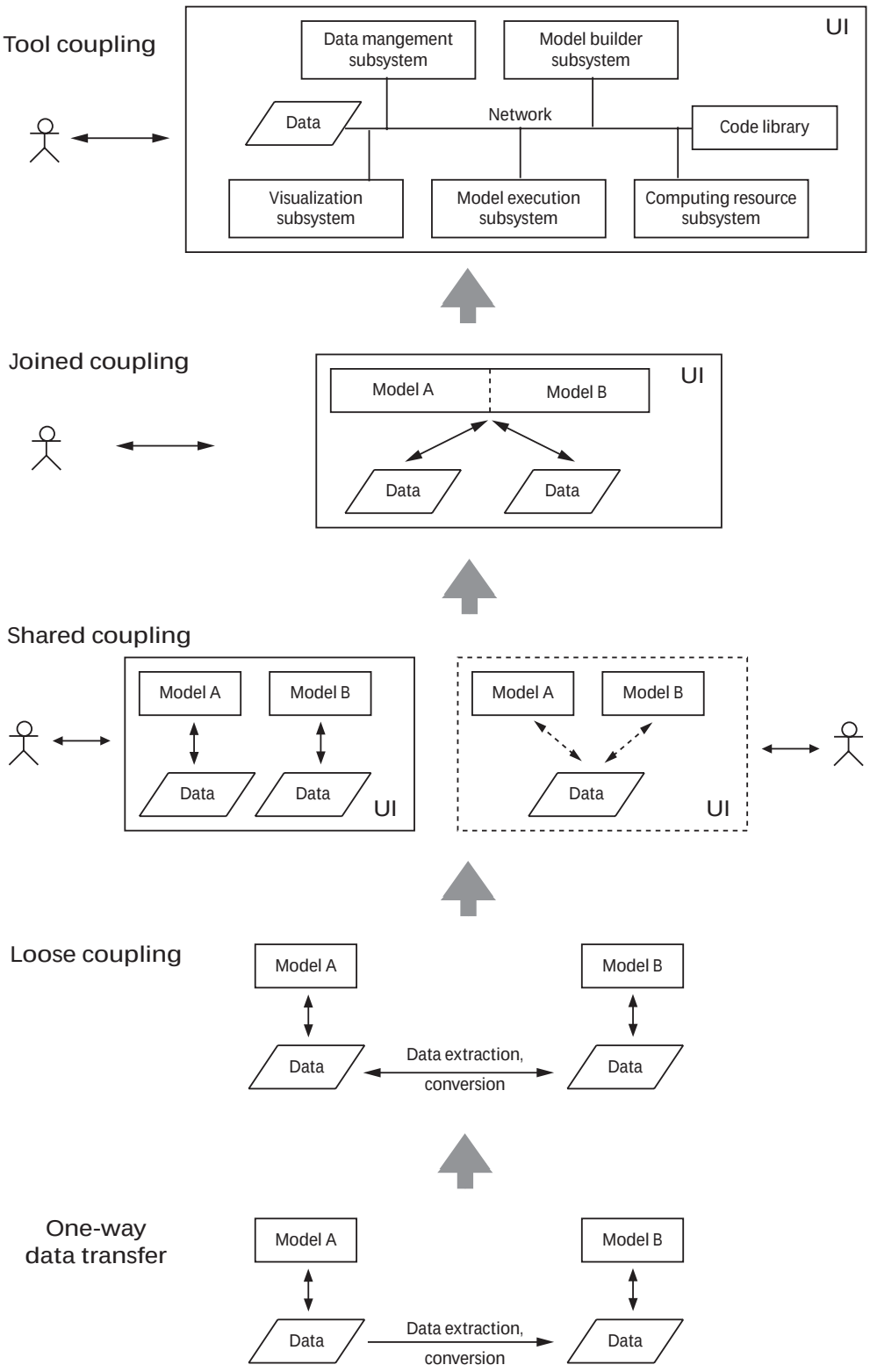

Figure 3-25 The ‘maturing typology’ of integration, after Brimicombe (2003), based on Brandmeyer and Karimi (1996).

'Joined coupling' exhibits both, a common UI and a common storage, either as *embedded coupling* (similar to 'embedded' in the typology illustrated in Figure 3-23), where one subsystem is embedded in the other, or as *integrated coupling*, where the subsystems are integrated as peers (similar to tightly coupled in the typology illustrated in Figure 3-23). Embedded coupling appears for example, when a modeling language is used to implement a simulation model within a GIS; integrated coupling is supported by interoperable services specified by the OGC. *'Tool coupling'* is considered to be the highest level of integration. It represents rather a framework comprising several subsystems, typically each dedicated to provide one specific service. For the ease of the user, a common UI may be provided wrapping all subsystems, which can be distributed over a network. Additionally, a shared data storage may be provided. Both, shared and joined coupling are used within the tool coupling framework.

Linkage-centered typology

Taking a more abstract, software engineering-oriented perspective, Westervelt and Shapiro (2000) suggested a classification of combination approaches of TSS. A dimension of this classification was the question whether a non-expert user can use the different combination approaches to solve interdisciplinary problems, for example, in a SDSS. The typology can be perceived as largely domain independent (i.e. not only applicable to combinations of TSS) due to its interdisciplinary aspect and due to the software architecture-oriented focus (Wytzisk, 2003). The typology consists of five types.

'Stand-alone subsystems' cannot be assumed to have any commonalities in terms of data structure, UI, programming language, operating system, inter-process communication, data file format, and output format (cf. Figure 3-26).

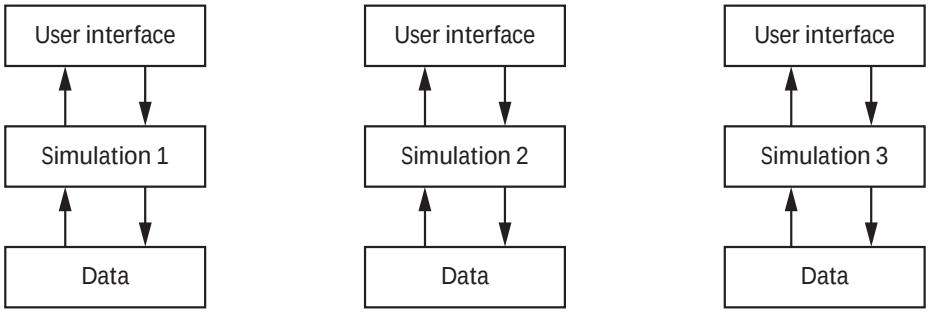

Figure 3-26 Stand-alone simulation models, after Westervelt and Shapiro (2000).

The '*shared assets and procedures'* class address subsystems operating in a common environment, which means the subsystems use the same computer hardware resources and the same operating system. Moreover, the subsystem inputs (e.g. parameter and input data files) are extracted automatically from a common data and software environment (e.g. a GIS). The systems share a common look and feel in their interfaces; this is symbolized with the grey band in Figure 3-27. The relative consistency of the UIs and data sources facilitates the establishment of a common UI and data source. An excellent example is the GRASS (cf. subsection 3.3.4), where a set of routines is arranged in libraries providing a common UI, access to data, visualization of information, and data processing. Benefits of this approach are an increased user learning curve due to the consistency of the UI and the data structure. To reach the required consistency starting from independent legacy subsystems can require a significant effort (Westervelt and Shapiro, 2000).

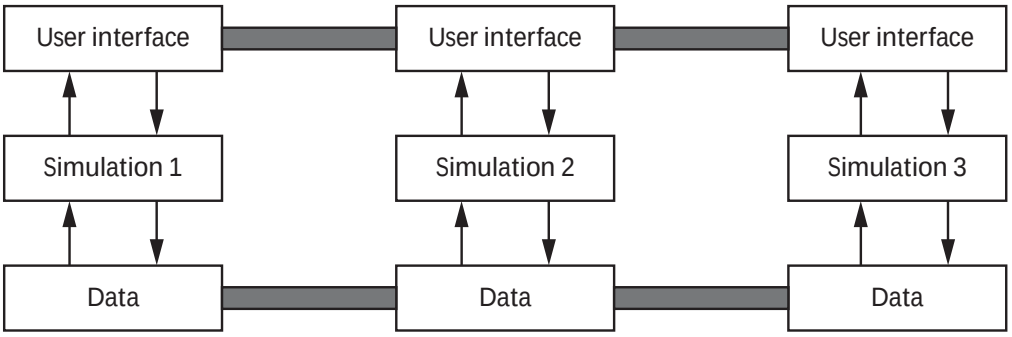

Figure 3-27 Shared assets and procedures, after Westervelt and Shapiro (2000).

While in the 'shared assets and procedures'-approach subsystems share a common environment, in '*linkages between programs*' autonomous subsystems are addressed. The output of a subsystem is automatically converted to serve as input to other subsystems. This is shown in Figure 3-28. This mechanism is similar to pipe constructs in UNIX, where output of a program is sent directly to the input of another program without any saving to disk. Examples of linked commercial programs running on the same platform are the ERDAS image processing systems[56] and the ESRI ModelBuilder[57] software. An advantage here is the eased linking of legacy system, since no changes to their code is necessary (Westervelt and Shapiro, 2000). The 'linkages between programs' approach links a set of subsystems allowing communication to flow from one subsystem to the other. Feedback loops are not supported by this mechanism.

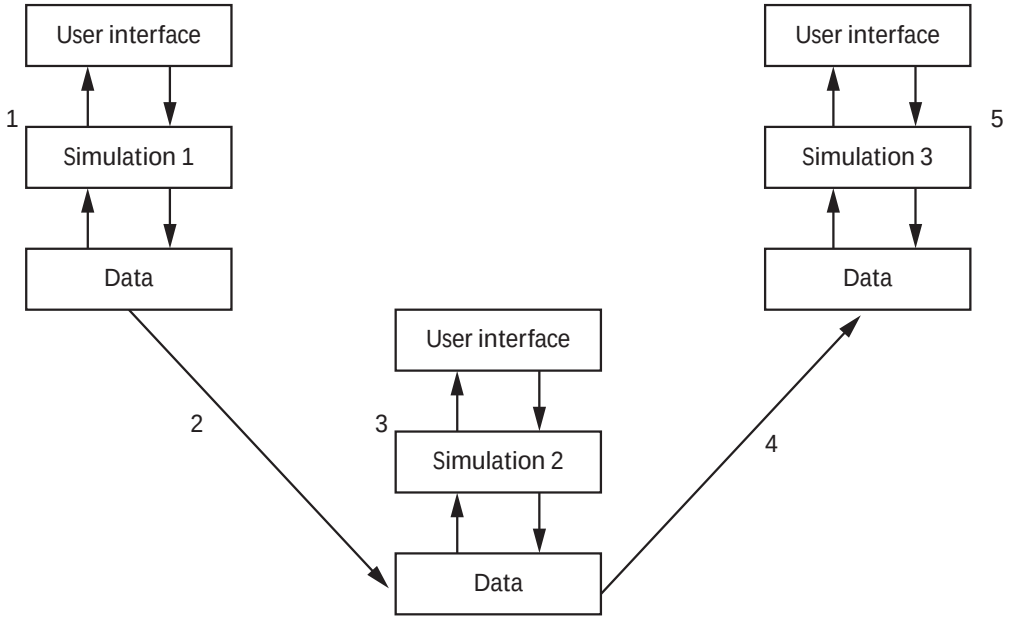

Figure 3-28 Linkages between programs: The numbers indicate the sequence, the arrows illustrate that output of one program is the input of the next program, after Westervelt and Shapiro (2000).

[56] http://www.gi.leica-geosystems.com/Products/Imagine/
[57] http://www.esri.com/software/arcgis/about/desktop.html#modelbuilder

In '*dynamic linkages between programs',* all subsystem use a common execution environment, as indicated in Figure 3-29 by the grey band, providing common communication and synchronization mechanisms. This supports both the linear coupling of models as well as the introduction of feedback loops between the subsystems. Inter-process communication can be established by time stepping, where each subsystem is only run for a short period and after that, initialization of each subsystem can be adapted. This approach minimizes changes to the legacy code, but becomes computationally inefficient. Another possibility is using a common synchronization and communication framework, which probably requires significant changes to the legacy systems. HLA is a representative for this solution. The '*linkages with distributed objects'* approach extends the 'dynamic linkages between programs'-approach by using distributed component models to be applied in a distributed software environment. HLA can also serve as an example here.

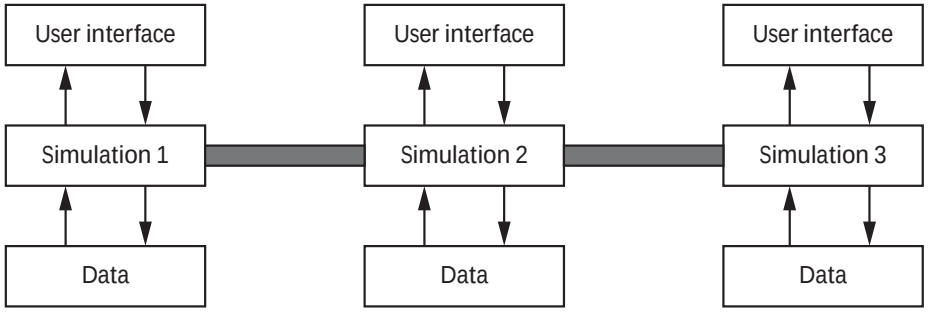

Figure 3-29 Dynamic linkages between programs, after Westervelt and Shapiro (2000).

The overview in Table 3-6 illustrates the tradeoffs of the different approaches. Moving down the list of approaches the effort required to adapt legacy systems, the degree of model integration, and the interoperability of the subsystems is increased.

Approach	Use legacy software	Distributed processing	Inter-system feedback loops
Stand-alone programs	a	c	d
Shared assets and procedures	b	c	c
Linkages between processes	c	b	c
Dynamic linkages between processes	c	b	a
Linkages with distributed objects	c	a	a

Table 3-6 Overview of approaches comparing the degree to which the objectives (listed in the columns) are met, ranging from 'a' meaning that the approach supports the objective to 'd' meaning that the objective is not supported by the corresponding approach, after Westervelt and Shapiro (2000).

4 Synthesis and research approach

The key motivation of the IPODLAS project originates from deficiencies of current concepts and tools to comprehensively handle natural processes affecting the landscape. Part of the IPODLAS project, this thesis presents an approach to represent spatiotemporal and cross-scale processes in a holistic way by combining functionality of the three domains TSS, VR, and GIS. Applying a structured software development approach, common concepts from software architecture, recent advances in network technology, and the move of applications towards interoperability have been exploited to develop concepts and software systmes which take advantage of the combined usage of applications of TSS, VR, and GIS.

This is a short, but crucial chapter: It tries to gather arguments brought forward in the chapters 2 and 3, which are important for the development of the IPODLAS framework, and to derive the motivation and roadmap of the thesis. The section 4.1 condenses crucial issues of a combination of the three domains, the IPODLAS approach, and issues of interoperability relevant for IPODLAS system. Section 4.2 explains the research approach, which applies the pivotal issues identified in section 4.1 for the development of the IPODLAS framework. The relevance of IPODLAS framework for the domain GIS and the relations between the research approach and the research questions (specified in chapter 1) are detailed in section 4.3 and section 4.4, respectively.

4.1 Synthesis

Combinations of TSS, VR, and GIS

Processes taking place in nature exhibit most often spatial and temporal patterns and are rarely confined to one scale, but spread over several scales. To understand such processes, a holistic representation with respect to their spatial, temporal, and cross-scale nature is required (Pang and Shi, 2002). Since existing technologies cannot provide the whole range of required functionality or only in a restricted manner, the combination of functionality and knowledge from different domains with partially complementary strengths and weaknesses can be a promising approach (Brimicombe, 2003; Fedra, 1993; Fedra, 1996). TSS provide functionality for temporal aspects of the phenomena of interest, GIS contribute its spatial capabilities, and VR offers visualization and the main user interaction functionality.

There is ample research on combining TSS and GIS (Bennett, 1997; Brimicombe, 2003; De Vasconcelos et al., 2002; Fedra, 1993) and VR and GIS (Huang et al., 2001; Pajarola and Widmayer, 2001); particularly in planning there is some experience in combining TSS and VR. However, a considerabely smaller number of projects have applications of all three domains in one system (Wang, 2004). Problems of combinations of TSS, VR, and GIS originate from their respective foci on different aspects of the phenomena of interest. While the view of the world in VR and GIS is static and applies to two or three (spatial) dimensions, TSS have a dynamic, process-oriented perception of the world. As the data model is a simplified model of the phenomena of interest, the deviating notions of the world in TSS, VR, and GIS propagate to different data models (Aspinall and Pearson, 2000; Bennett, 1997; Fedra, 1993; Fedra, 1996), different storages approaches (Brimicombe, 2003), and different input/output techniques. Since interoperability standards of the three domains also employ the same deviating views of the world, combinations of the standards also exhibit the same shortcomings (Bernard, 2001; Wytzisk, 2003).

The IPODLAS approach

The IPODLAS approach specifies *case studies*, each providing data and simulation models. The case studies are situated in an alpine environment focusing on different aspects so as to provide a broad and diverse range of user requirements. The case studies provide realistic data and simulation models on different scales describing spatiotemporal and cross-scale processes and acting as testbeds for implementations of prototypes (Allgöwer et al., 2003; Isenegger et al., 2005). Applying the 'Unified Process' (UP) (Jacobson et al., 1999), *use cases* situated in the case studies are used to explicitly define the requirements of users in a formal and reproducible way and to break down the complexity of the development process into Mini-projects (cf. section 2.1). Additionally, *sequenced action lists* and *functionality lists* have been generated to specify user interaction and the provision of functionality, respectively (Isenegger et al., 2005).

Interoperability

In the previous chapter it has been shown that interoperability is the ability of a system to cooperate with others (Ghezzi et al., 2003). Subsystems which comply with relevant interoperability standards of their domain share a common view of the phenomena of interest and thus increase the applicability of standard means to exchange information across systems and domains (Wytzisk, 2003). Traditionally, GIS have been closed, isolated applications with heterogeneous structures and functionality (Preston et al., 2003). The growing acceptance of interoperability standards, such as standards from the OGC and ISO/TC 211 (cf. subsection 3.3.2), led to an increased compliance of GIS with interoperable GIS specifications. An example for this is the application of GML as an input or output format of GIS. For TSS, the dominant interoperability approach HLA (cf. subsection 3.1.2) provides similar standardization possibilites (Wytzisk, 2003), whereas in the domain of VR no dominant standard of comparable relevance exists (Van Dam et al., 2000). However the standardization approaches of TSS and VR are not in the focus of the thesis.

Furthermore, the interoperable usage of applications from the three domains is enhanced by the provision of an application programming interface (API) of a major programming language, such as Modula-2, C++, Java, or Python. Thus, the access to functionality of an application from an arbitrary environment is facilitated. The concepts and the software architecture of the IPODLAS system rely mainly on standard, non-proprietary software to ensure a broad applicability and not to be dependent on a specific subsystem or software environment. Modular systems limit interdependencies between the subsystems through minimal interfaces (Ghezzi et al., 2003). To achieve openness the IPODLAS system must exhibit a modular structure; this supports, for instance, the replacement of a specific application with another. Recent developments, such as the emergence of networked applications and the growing maturity and acceptance of software interoperability standards (cf. section 2.3), support the combined application of subsystems. In parallel, the move from stand-alone applications towards network-centric approaches supports the combined usage of applications (Batty, 1999; Brimicombe, 2003), particularly in terms of exchanging data and accessing functionality across system borders.

4.2 Research approach

The key concept of the IPODLAS project is bringing together the three domains TSS, VR, and GIS allowing a spatiotemporal, cross-scale handling of processes affecting the landscape (Allgöwer et al., 2001; Isenegger et al., 2005). This thesis situated in the IPODLAS project applies a *threefold approach* to develop the IPODLAS framework, which can support a comprehensive representation of spatiotemporal, cross-scale processes.

The components of the threefold research approach — combined usage of functionalities of the three domains, the methodology IPODLAS approach, and recent advances in software technology and interoperability standards — are described in the subsection 4.2.1, 4.2.2, and 4.2.3, respectively. The research approach gathers important issues for the development of IPODLAS framework identified in section 4.1 and illustrates their significance and application.

4.2.1 Combination of the three domains

The combination of subsystems of the three domains provides the capability to handle spatiotemporal, cross-scale processes in a holistic manner. By exploiting the particular strengths and by avoiding the specific weaknesses of the subsystems of the three domains, the IPODLAS system bears the promise of gaining synergies and mutual support, which may lead to a cross-fertilization of the corresponding domains (Isenegger et al., 2005). The core of the presented concept is the design activity of a system whose subsystems provide services, in that sense, that answers are provided to requests coming from other subsystems. The IPODLAS system offers *services*, which focus on solving specific tasks required to deal with spatiotemporal, cross-scale processes described in the framework of the case studies. When a subsystem lacks a certain functionality, it can — if available within the IPODLAS system — request the required functionality provided by another subsystem. The service-oriented nature of the IPODLAS system in terms of subsystems offering functionalities supports the use of the best of the three domains regarding flexibility and capability. However, the services provided by the IPODLAS system are not web services (cf. section 2.3.2); for instance, they don't use HTTP as transport protocol and they are not published in the network. The subsystems can benefit from functionality provided by other subsystems without having to share a completely congruent understanding of the phenomenon of interest. The provision of functionality as services across subsystem borders is facilitated if the subsystems are not designed as standalone applications, but to be integrated into a networked environment. The growing acceptance of and the compliance with interoperability standards in the respective domains, such as GML, supports the interoperability of subsystems within the IPODLAS system.

4.2.2 The IPODLAS approach

The methodology applied to derive user requirements and to design the IPODLAS system supports a formal and reproducible way of development[1]. In Isenegger et al. (2005) this methodology is called the *IPODLAS approach*. The framework of case studies consists of three case studies from different domains selected to capture a broad range of requirements. The Larch Bud Moth (LBM) case study represents insect population dynamics, the wildland fire modeling (WLF) case study is an example of an abiotic process, wildland fire visualization (WFV) represents a case study where the focus is on 3-D photorealistic visualization of a spatiotemporal, cross-scale process. These case studies were chosen to include both spatial and temporal aspects and to offer models and associated data across several scales. The application of simulation models on different scales enables the IPODLAS system to represent cross-scale processes in a more appropriate way than only using one simulation model for all scales. The UP has been used to derive user requirements and to develop the IPODLAS system in a formal and reproducible way. The use case model consists of a definition of the users and the description of all use cases. Each use case is first described in prose, which defines the

[1] This paragraph is based on section 4.1 and 4.2 of Isenegger et al. (2005).

particular intentions of the user and the interactions of the user with the system. The prose text then is refined in a sequenced action list, where the interaction of the user with the system is defined step by step by specifying the input of the user and the response of the system. All functionality specified in the use cases is compiled in a list of functionality defining the range of functionality the IPODLAS system must provide. Additionally, the sequence of interaction and the divsion of labor on the subsystem level is specified. Several use cases have been implemented in different prototypes providing new insights and acting as proof of concept (Isenegger et al., 2005). The IPODLAS approach is described in greater detail in chapter 5.

4.2.3 Interoperability

The growing compliance with interoperability standards and the move from stand-alone to networked applications supports embedded usage of the legacy systems used in the IPODLAS system (RAMSES, VTP, and GRASS; cf. subsections 3.1.4, 3.2.4, and 3.3.4, respectively). The provision of an API of a modern programming language facilitates the embedding of applications into a software environment. RAMSES comes with a Modula2 API, VTP provides a C++ API, and GRASS supports access using C++ and shell scripts. The IPODLAS system exhibits a modular structure by limiting the dependencies of the subystems through well-defined, minimal interfaces. In the final IPODLAS prototype (cf. chapter 7), all communication between the subystems is accomplished via the sockets interface. The information exchange is established by sending messages carrying XML encoded information. Beneficial features of networked technologies and of the compliance of applications with software interoperability standards are applied by the IPODLAS system to establish a software environment, where subsystems from the domains TSS, VR, and GIS can cooperate. For instance, the final IPODLAS system applies the internet socket interface for accessing the legacy systems and to exchange information. Exchanged data and control information used to synchronize the IPODLAS system and to request functionality is encoded using XML. Exploiting potentials of network technologies and the subsystems API, functionalities of all subsystems within the IPODLAS system can be accessed from any subsystem. Thus, the IPODLAS system supports the seamless usage of functionality, data, and simulation models, which are provided by the individual subsystems.

4.3 IPODLAS and GIS

The thesis describes a methodology to derive the requirements that a system such as IPODLAS pose to its embedded GIS subsystem. Furthermore, a software system is specified which enables GIS (together with TSS and VR) to handle spatiotemporal, cross-scale processes in a holistic manner. Preliminary prototypes of IPODLAS, which implements different aspects of spatiotemporal processes, are described in the chapter 6, while the final prototype is detailed in chapter 7.

The use cases situated in the framework of the case studies and the resultant functionality listings define the GIS functionality required to satisfy spatial representation and analysis issues of a system such as IPODLAS. In addition, the functionality listings derived from the use cases define the information exchange between the subsystems, which is required to develop specifcations of interfaces between the respective subsystems. The framework of the case studies, the use cases, and the derived functionality listings are described in chapter 5.

The combination of subsystems from TSS, VR, and GIS in the sense of ‘tool coupling’ of the ‘maturing typology’ (cf. subsection 3.4.2) of Brimicombe (2003) in a networked environment demonstrates how GIS can participate in handling spatiotemporal, cross-scale processes. The final IPODLAS system coordinates the

collaboration of the subsystems to fulfill a task required by a user. The respective subsystems provide their functionality to be used by the IPODLAS system. So if a task requires any spatial functionality, the IPODLAS system can access the corresponding GIS functionality via the specified interfaces from the GIS subsystem.

Applying an interoperability standard of the GIS domain, GML is used in an preliminary IPODLAS prototype for the information exchange and storage of spatiotemporal data (cf. subsection 6.2.2). Using the dynamic features of GML 3, temporal infomarion coming from the TSS can be integrated with the spatial information in a single data representation.

4.4 Research questions

The goal of the thesis is the development of the *IPODLAS framework* which embraces methods and concepts supporting the represention of spatiotemporal and cross-scale environmental processes in a holistic manner. The *development methodology IPODLAS approach* specifies usage scenarios of the IPODLAS system and thus helps to break down the complexity of the development process and specify the collaboration and contributions of different subsystems. The methodology is applied to develop *concepts* and a *software architecture* for the *IPODLAS system* which have to combine functionality of different legacy systems. The Table 4-1 elaborates how the research approach explained in section 4.2 is applied to provide answers to the research questions specified in chapter 1.

Research question from chapter 1	**Resources from the research approach (cf. section 4.2) to answer the research question**
1. What is the appropriate development methodology to gather the full range of user requirements and system constraints for the development of a system such as IPODLAS?	The IPODLAS approach (cf. subsection 4.2.2) specifies a methodology which supports the specification of requirements and constraints defined in research question 1. The IPODLAS approach is described in greater detail in chapter 5.
2. Is the standard GIS functionality sufficient to support the requirement of a system such as IPODLAS?	The IPODLAS approach consisting of use cases allows the specification of a functionality listings describing the functionality required in the IPODLAS system (outlined section 4.3 and elaborated in section 5.3)
3. What are suitable concepts and architectures for a software system to meet the goals of the IPODLAS project, which are to develop a framework combining the three domains, GIS, VR, and TSS to facilitate the joint seamless usage of functionality, data, and models?	Potentials and benefits of the three domains (cf. subsection 4.2.1) and of advances in software technology (cf. subsection 4.2.3) are exploited using the IPODLAS approach (cf. subsection 4.2.2) to specify maturing concepts and software architectures for the IPODLAS system. These architectures are implemented in several prototypes described in the chapters 6 and 7.

Table 4-1 Relations between the research questions specified in chapter 1 and the research approach defined in section 4.2.

Part II

The tripartite nature of the thesis is applied to separate the experimental research part from the theorectical considerations in Part I and from the evaluation in Part III. In Part II different aspects of the IPODLAS framework development described in the chapters 5, 6, and 7 are bundled in one coherent conceptual structure.

5 The IPODLAS approach

The IPODLAS approach is a crucial part of the research approach of this thesis presented in in section 4.2. The methodology called IPODLAS approach is applied to develop the IPODLAS system. The IPODLAS approach encompasses the case studies and the use cases situated therein, which have been developped using the structured software development process UP (Jacobson et al., 1999) (cf. section 2.1). Since the methodology applied to develop the IPODLAS system is seen as one of the result, it is presented in this second part of the thesis.

Section 5.1 details the case studies and the simulation models they provide. An overview of the use cases situated in the case studies and two concrete examples of how to get from the prose description of a use case to the identification of the required functionality is the subject of the section 5.2. In section 5.3 the requirements of the IPODLAS system concerning GIS functionality are derived in an exemplary way.

5.1 Case studies

Three case studies have been applied in the IPODLAS framework to develop IPODLAS system; they provide data and simulation models. The case studies and associated simulation models are described in greater detail in Price et al. (2003). As outlined in chapter 1, in the IPODLAS project three subprojects are conducted, each focusing on one subsystem and one case study. The application of the TSS subsystem on the case study LBM in the IPODLAS project is described in greater detail in Price (2005), Price et al. (in press), and Price et al. (submitted). The case study WLF and the usage of GIS is topic of this thesis and of Isenegger et al. (2005). In Wu et al. (submitted) and Wu et al. (accepted) the application of VR is detailed in the case study LWV.

5.1.1 Larch Bud Moth (LBM)

The *Larch Bud Moth* (LBM) (*Zeiraphera diniana* GN.; Lep., Torticidae) causes defoliation of larch trees (*Larix decidua*) across the entire Alpine Arc every eight to ten years (Baltensweiler and Fischlin, 1988). Larch trees which have been defoliated exhibit an unattractive brown colour during summer time and produce after defoliation a set of foliage with reduced nutritional value for the LBMs; this effect can remain for several years (Fischlin, 1982; Price, 2005). Several hypotheses have been developped to describe the temporal dynamics of the LBM. In the IPODLAS framework the 'food-quality hypothesis' is applied, which correlates the reduced nutritional value of defoliated larch trees with occurence of LBMs (Fischlin, 1982; Fischlin and Baltensweiler, 1979; Price, 2005). While the temporal patterns of the LBM population cycles have been researched intensively, the spatial patterns have gained less attention (Price, 2005). Some studies describe traveling waves of LBM across the Alpine Arc (Bjornstad et al., 2002) probably due to wind-driven dispersal in conjunction with a gradient in habitat quality. Figure 5-1 shows that the LBM cycles are in close synchronicity with one another at the scale of the Upper Engadine valley (Fischlin, 1982; Fischlin, 1983). However, there are some exceptions where migration is restricted by topography (Price, 2005). At the valley scale,

local dispersal is dependent on wind conditions, which may vary significantely over small areas due to rugged terrain (Baltensweiler and Rubli, 1999). Several simulation models describing the LBM dynamics are available to the IPODLAS framework (cf. 5.1.4), each modeling the LBM behavior on a different scale.

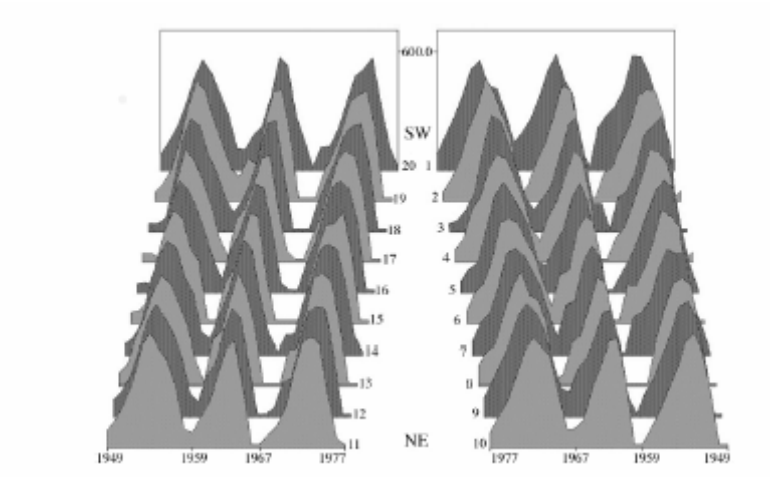

Figure 5-1 Observed LBM larval densities in the 20 sites of the Upper Engadine valley for the period 1949 - 1977 (Fischlin, 1982). This figure schematically depicts a view of the Upper Engadine valley, where the observer looks from north-east to the south-west having larval densities (of the period 1949 to 1977) of the sites 1 to 10 on the right side and the larval densities of the sites 11 to 20 on the left.

Research area

In Price (2005) the LBM dynamics is modeled at the spatial extent of the Upper Engadine valley, which is a mountain valley in the Swiss Alps located in the south-eastern part of Switzerland (cf. Figure 5-2). The research areas are situated at an elevation above 1600 m above sea level (Fischlin, 1982). The three LBM dynamics simulation models LBM-M8, LBM-L10, respectivley LBM-M11 (cf.Table 5-1) are applied to the same spatial extent (i.e the Upper Engadine valley), but assume different spatial resolutions:

1. the *valley*: The LBM-M8 treats the entire Upper Engadine valley as one point in space.
2. the *sites*: In LBM-M10 the forest cover in the Upper Engadine valley is divided into 20 sites of an average area of 3.7 km^2, which are homogeneous with respect to altitude, forest type, and aspect. For each site a center is defined located about at the center of gravity of the respective site (Fischlin, 1982).
3. the *forest compartments*: In LBM-M11 the forest cover in the Upper Engadine valley is divided into 420 forest compartments of an average area of 25 hectares.

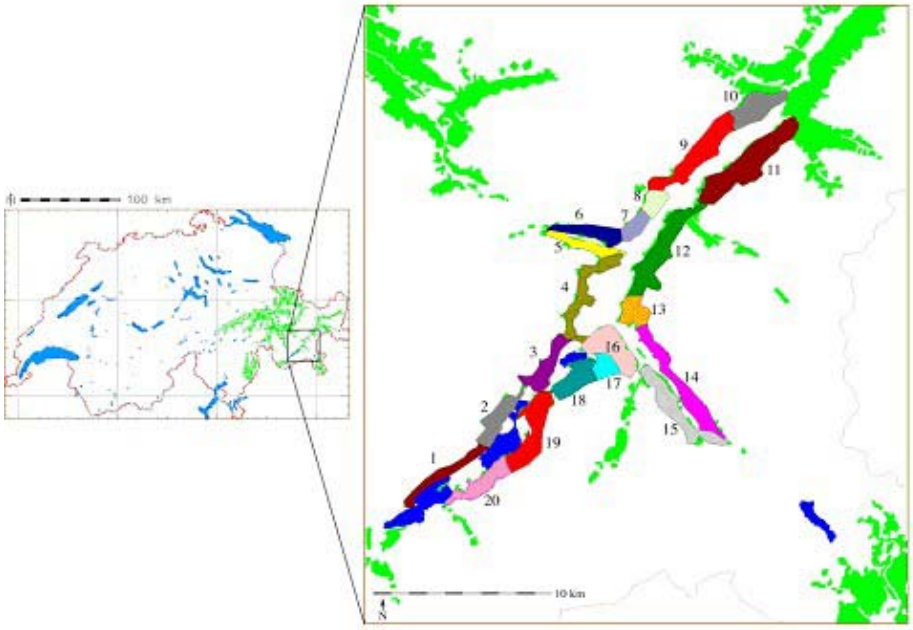

Figure 5-2 The left picture shows the location of the Upper Engadine Valley, Switzerland. The picture on the right side illustrates the location of the 20 sites within the Upper Engadine valley. The forest compartments define a division of the same forested area as the sites exhibiting a higher spatial resolution (Price, 2005).

5.1.2 Wildland fire (WLF)[1]

Wildland fire is dependent on wind, topography, fuel, humidity, and other factors. The *surface fire* is the most common wildland fire type. It occurs often at the beginning of a fire. As Figure 5-3 illustrates, surface fires can cause *crown fires*, if the intensity of a surface fire reaches a certain threshold dependent on the height of the canopy, the humidity of leaves, and the density of the canopy. *Fire spotting* happens when *burning particles* are transported by convective air currents. This fire type can be important for crossing barriers (Jecklin and Schöb, 1993).

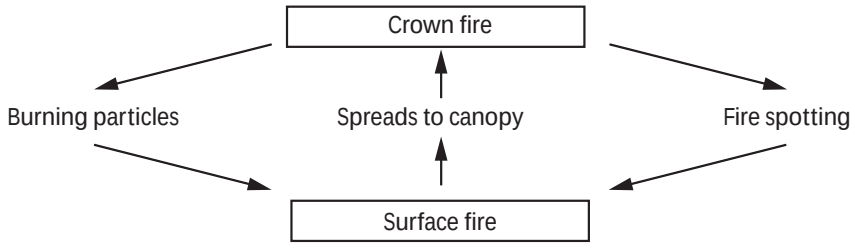

Figure 5-3 Types of forest fire and fire propagation processes, after Jecklin and Schöb (1993) and Schöning (1996).

Common simulation models describing the behavior of WLF are shown in Figure 5-4. A conventional approach to simulate the spread of a WLF is to combine models describing the shape of a fire spread and models simulating the local behavior of fire (Finney, 2004).

[1] This subsection is based on chapter 5 of Price et al. (2003).

The *shapes of fires* are often represented as ellipsoids assuming homogeneous conditions of fuels, weather, and topography (Alexander, 1985; Anderson, 1983) with distortions of the ellipse towards higher wind speed and uphill (Alexander, 1985).

The *local fire behavior* models can be divided into three categories. The *empirical models* are based on statistical analysis of past fires. While complex fuel properties are implicitly treated in such models, the applicability of the results is strongly correlated with the conditions of tests compared to conditions of real wildland fires, mainly concerning types of fuels and the wind field (Rothermel, 1972). *Semi-empirical models* apply physical laws of energy conservation to calculate the spread parameters from measurable fire parameters. On the one hand, this model approach generates measurable parameters and a larger validity range of the model. On the other hand, the simulation of natural conditions, due to their complexity, is incomplete and applies better to smaller fires (Rothermel, 1972). Since a wildland fire is a very complex process, there exist to date no complete *physical models* that describe a WLF exhaustively. Most physical models assume some restrictions, for example only surface fires are addressed, the fuel is assumed to be homogeneous, and/or the flame front is approximated to be a rectangle with limited height and constant temperature. The energy balance of a volume element is then calculated using differential equations. An advantage of this approach is that it is still applicable when spread data is missing. On the other hand there is no complete physical model of the heterogeneity of fuel and the complex environmental conditions are difficult to consider (Schöning, 1996).

Simulation models describing different types of wildland fires on distinct scales are listed in Table 5-1 in the line WLF; all WLF simulation model applied in the IPODLAS framework base on the simulation model of Rothermel (1972). Typical input data for WLF simulation models are *slope*, *aspect*, *wind direction* and *speed*, *fuel models*, and *fuel moisture*. A fuel model is an abstraction of a set of fuelbed inputs required in WLF simulation models. The WLF simulation models *r.spread* (Xu, 1994), which is the WLF spread simulation in GRASS, and *FARSITE* (Finney, 1998) (cf. Table 5-1) apply the National Forest Fire Laboratory (NFFL) fuel models (Fischer, 1982). *SPARKS* (Schöning, 1996) additionally use specific Swiss fuel models (Allgöwer et al., 1998; Harvey et al., 1997).

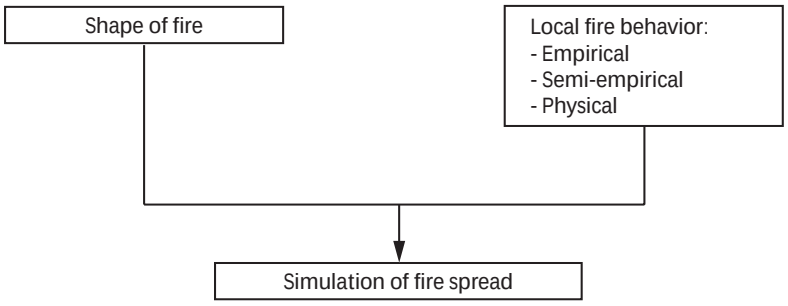

Figure 5-4 Overview of common wildland fire simulation models, after Schöning (1996).

The research area

The Swiss National Park (SNP) is located in the Lower Engadine, a mountain valley in the Swiss Alps located in the south-eastern part of Switzerland. This region can be fire prone, since in summertime the climate can be rather dry. The lower parts of the SNP are mainly covered with forest, with most of the trees being conifers. The remainder of the park, mainly the higher areas, are either meadows or stony, rocky terrain. Most of the WLF simulation models in this project are applied in the SNP. However, when WLF

simulation is combined with LBM simulation models, they are applied in the Upper Engadine study area.

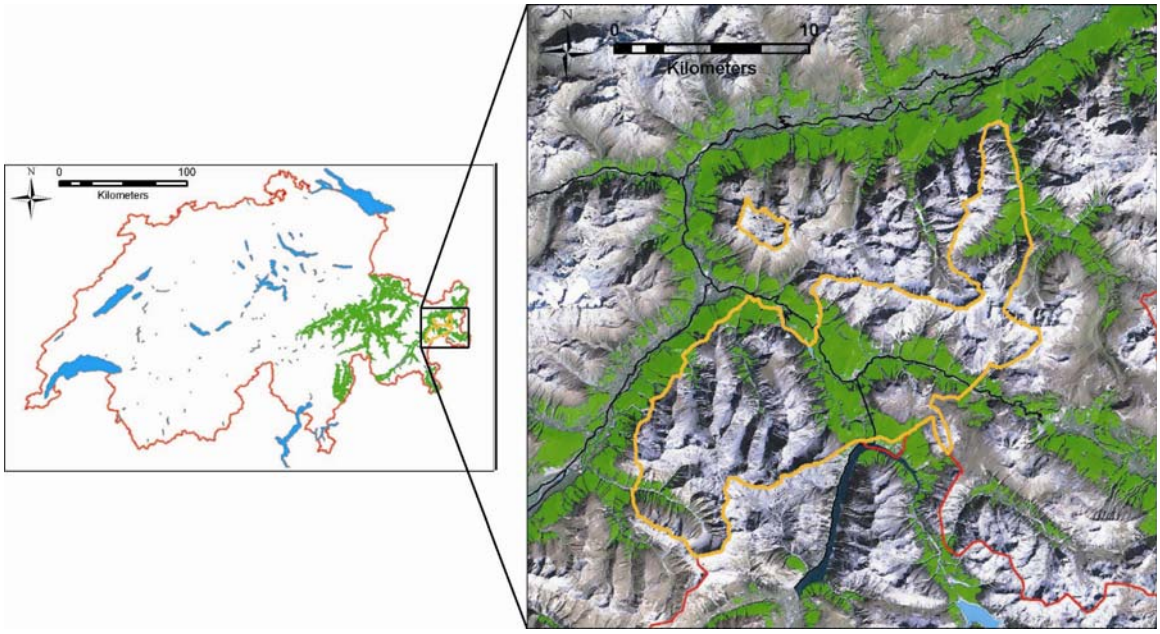

Figure 5-5 The left picture shows the location of the Swiss National Park (SNP), Switzerland. In the picture on the right side, the borders of the SNP are displayed in orange, while the borderline of Switzerland is represented in red.

5.1.3 Larch Bud Moth (LBM) and Wildland fire (WLF) visualization (LWV)

The case studies LBM and WLF deal with environmental processes, whose observation may produce spatiotemporal and cross-scale — i.e. complex — data. Visualization of complex data can significantly enhance the user's understanding of the data and hence, of the underlying processes (Biegger, 2004; Kraak et al., 1999). In both case studies, LBM and WLF, large data sets, e.g. DEMs and the satellite images covering the DEMs, have to be rendered. In addition, both case studies require the visualization of spatiotemporal data, e.g. the output of simulation models, which is located somewhere in the geographic space. These requirements challenge the visualization capabilities of today's common computers and require intelligent and effcient visualization approaches (Wu et al., accepted).

LBMs in the Upper Engadine valley are modeled to potentially migrate from each site to each other site. In the case study LWV the migrating LBMs are visualized as particles forming clouds starting from one site and fly to their target site(s) (cf. Figure 5-6). LBM clouds are dynamic because they are composed of dynamic objects which do not keep their relative positions. Challenges in cloud rendering are the optical properties along light paths trough the cloud volume and the complex scattering of light within the medium (Wu et al., accepted). Color changes of the larch foliage resultant from LBM infestation are visualized using polygon-based representations of the infested sites, which are placed above the virtual terrain (Wu et al., accepted) (cf. Figure 5-7).

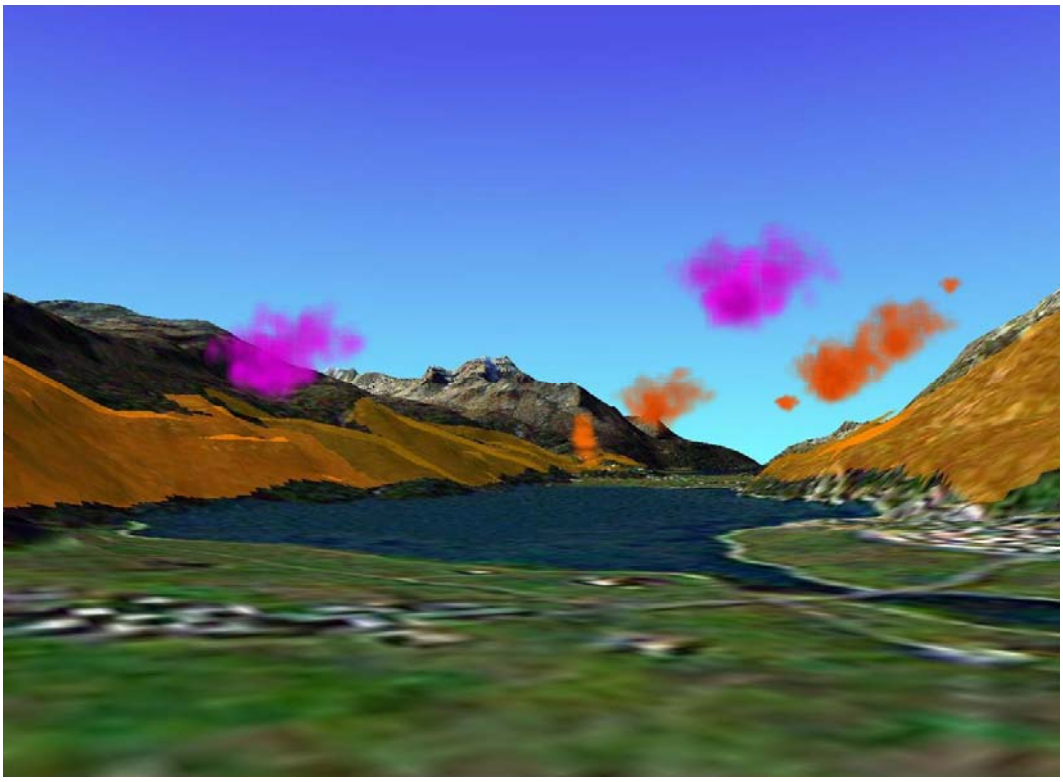

Figure 5-6 Screenshot of a VTP visualization of LBM migrating in the Upper Engadine. The LBMs are represented as particles forming clouds, which migrate from the starting site to the target sites. The LBM clouds are colored according to the site they origin from (Image courtesy of Y. Wu).

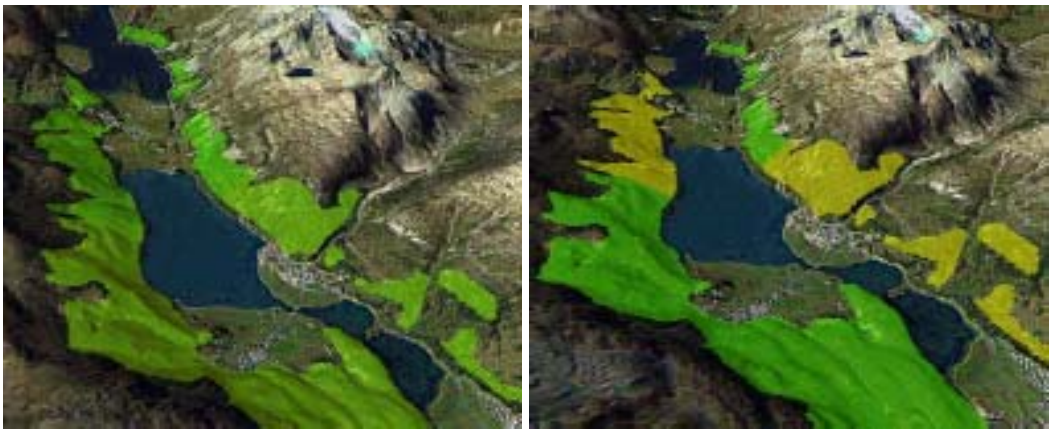

Figure 5-7 Screenshot of a VTP visualization of defoliation of sites due to LBM infestation. The green color in the left picture represents no defoliation. In the left picture trees in sites colored in yellow exhibit a medium defoliation rate (Wu et al., accepted), Image courtesy of Y. Wu.

For the *visualization of wildland fire*, besides the highly dynamic nature of wildland fires, also the semi-transparent characteristics of gaseous phenomena such as flames and smoke must be represented realistically. Turbulent flows of gaseous phenomena characterized by highly unsteady behavior can be simulated using computational fluid dynamics[2]. The crucial challenge in computer graphics when simulating gaseous phenomena is the modeling of turbulent flows in a visually convincing manner while maintaining interactive animation rates (Lamorlette and Foster, 2002) without loosing physical accuracy (Wu et al., submitted). In the LWV case study, this means, that the fire spread calculated by wildland fire simulation models (cf. subsection 5.1.4) must be visualized in a visually realistic manner and at a frame rate which allows realistic visualization[3]. In particular, the work accomplished in this case study focuses on the transitions of WLF visualization when moving across scales. For instance, starting at a small scale the flames of a burning branch of a tree are visualized. When the viewing distance between the fire and the observer is increased, i.e. the observer moves away from the fire, her/his focus shifts from individual flames to the visualization of an entire tree burning or even of a small stand of trees burning. At the largest scale, wildland fires at valley scale demand the visualization of a large fire extents with coarser resolutions. The LWV case study is applied to both research areas, the Upper Engadine valley and the SNP.

5.1.4 The case study framework

The case study-simulation model matrix consists of the case studies described in sections 5.1.1, 5.1.2, and 5.1.3. They were chosen to capture a broad range of requirements users may pose to a system like IPODLAS. All case studies include both, spatial and temporal aspects and offer models and associated data situated on all observed scales (Allgöwer et al., 2003; Isenegger et al., 2005) (cf. Table 5-1). The application of multiple simulation models on observed each scale allows a more accurate representation of cross-scale processes than using a single simulation model for all scales. Each member of the IPODLAS project team conducted projects primarily in one case study, but in each project functionality of each domain was involved. So, for instance, when the IPODLAS system is applied to simulate LBM at medium scale, TSS employs GIS functionality to support the calculation of spatially explicit LBM dynamics, while the user can investigate the visualization of the LBM migration results in the VR application.

[2] Computational fluid dynamics (CFD) is the use of computers to analyze problems in fluid dynamics (Anderson, 1994).

[3] The frame rate is the number of frames of an animation which are displayed every second. Typical values are 24 frames per second (fps) for movies; frame rates of 20 or more are usually considered as smooth (http://www.vterrain.org/Misc/glossary.html, accessed March 1, 2006).

	Applied model		
Case study	Small scale	Medium scale	Large scale
LBM (Larch Bud Moth)	*LBM-M8*: Local LBM dynamics, i.e. the Upper Engadine valley is treated as a homogeneous area with no spatial structure (Fischlin, 1982).	*LBM-M9*: combining M8 with migration within the valley, the Upper Engadine valley is divided into 20 sites (Fischlin, 1982).	*LBM-M11*: combining M8 with migration within the valley, the Upper Engadine valley is divided into 420 forest compartments (Price, 2005).
WLF (Wildland fire)	*Local Rothermel*: semi-empirical model describing fire spread in finite elements (Rothermel, 1972).	*SPARKS*: combines the Rothermel model with ellipsoid fire spread models covering surface fire (Schöning, 1996). *r.spread*: simulates elliptically anisotropic spread in GRASS based on the Rothermel model (Xu, 1994)	*FARSITE*: combines the Rothermel model with fire spread models covering surface and crown fire and fire spotting (Finney, 1998).
LWV (Larch Bud Moth and Wildland fire visualization)	Visualizing LBMs migrating at a single tree / Visualizing flames on a single branch.	Visualizing LBMs migrating within a forest stand / Visualizing a tree or a small stand of tree burning.	Visualizing LBMs migrating at the valley scale / Visualizing a wildland fire on valley scale.

Table 5-1 The case study-model matrix exhibiting the applied models of three case studies at three different scales[4], after Allgöwer et al. (2003) and Isenegger et al. (2005).

5.1.5 Listing and classifying the required functionality[5]

The use case model consisting of all use cases defines the range of the required functionalities that the IPODLAS system should entail in order to satisfy the requirements of the users specified in the various use cases. The functionalities recorded in the sequenced action lists form the basis of the functionality listings describing which functions have to be offered by which subsystem. In Table 5-2, the functionality is classified according to the estimated implementation effort of integrating this particular functionality into IPODLAS system. The classification of functionality together with the identification of the set of key use cases helps to discover the use cases with the greatest risks of failure.

Class	**Potential fulfilling of requirements**
1.	The required functionality is already implemented in one of the subsystems respectively legacy systems of the IPODLAS system.
2.	The required functionality is implemented in other software systems.
3.	A solution to offer the required functionality exists in the literature.
4.	No solution to offer the required functionality exists in the literature.

Table 5-2 Classification of functionality according to the estimated effort to integrate the respective functionality into the IPODLAS system (Isenegger et al., 2005).

[4] The WLF simulation model 'Local Rothermel' and the visualization models are not yet implemented on the respective subsystem.

[5] This section is based on section 4.3 of Isenegger et al (2005).

5.2 Use cases

The use case model consists of a definition of user types and the description of all use cases[6]. In the IPODLAS framework, two types of users are defined to cover diverse requirements, the pilot user and the expert user. The behavior of the *pilot user* is characterized by exploration; she/he "flies" through the virtual scenery and usually does not change any of the parameter settings but instead uses default configurations when running simulations. In contrast, the *expert user* is interested in the scientific capabilities of the system; she/he may want to change parameters of the particular subsystems and plug in new models.

Each use case is first described in a *prose text*, which defines the particular intentions and the interactions of the user with the system in order to reach the goal of the user described in the related use case. The prose text then is refined in a *sequenced action list*, where the interactions of the user with the system is defined step by step by specifying the input of the user and the response of the system. The data exchange between the subsystems within the IPODLAS system (e.g. type of data, valid range, etc.) is defined in the *data exchange table*.

The development of the use cases can be supported by the graphical definition of the GUI in a sequence of screenshot-like pictures. These pictures help to define the interaction options the system offers to the user at a given state of the system. The graphical definition of the GUI helps to specify the state the system is in and the functionality offered. For instance, in Figure 5-8 a state of the IPODLAS system within LBM simulation is specifed. The window in the background in Figure 5-8 is the navigation window of the IPODLAS system, where the user can "fly-through" the landscape, zoom in and out, and select phenomena of interest. On the menu bar, the user can select her/his topic of interest, which is currently restricted to LBM and WLF. When the user chooses LBM, she/he can configure her/his LBM session in the 'Larch bud moth configuratio'n window. If the user selects a simulation model, for instance 'Food quality and migration', she/he can access in the window 'Food quality and migration' the simulation parameters of this model and finally start the simulation.

Clearly confined use cases are beneficial when it comes to support an incremental development of the IPODLAS system. This facilitates the incremental and iterative addition of the functionalities of a new use case in the form of a Mini-project to the software system, which covers the functionality of the already implemented use cases. Among the set of use cases specified, the prospective users select the subset of the key use cases which entails the most important use cases.

[6] This section is based on section 4.2 of Isenegger et al (2005).

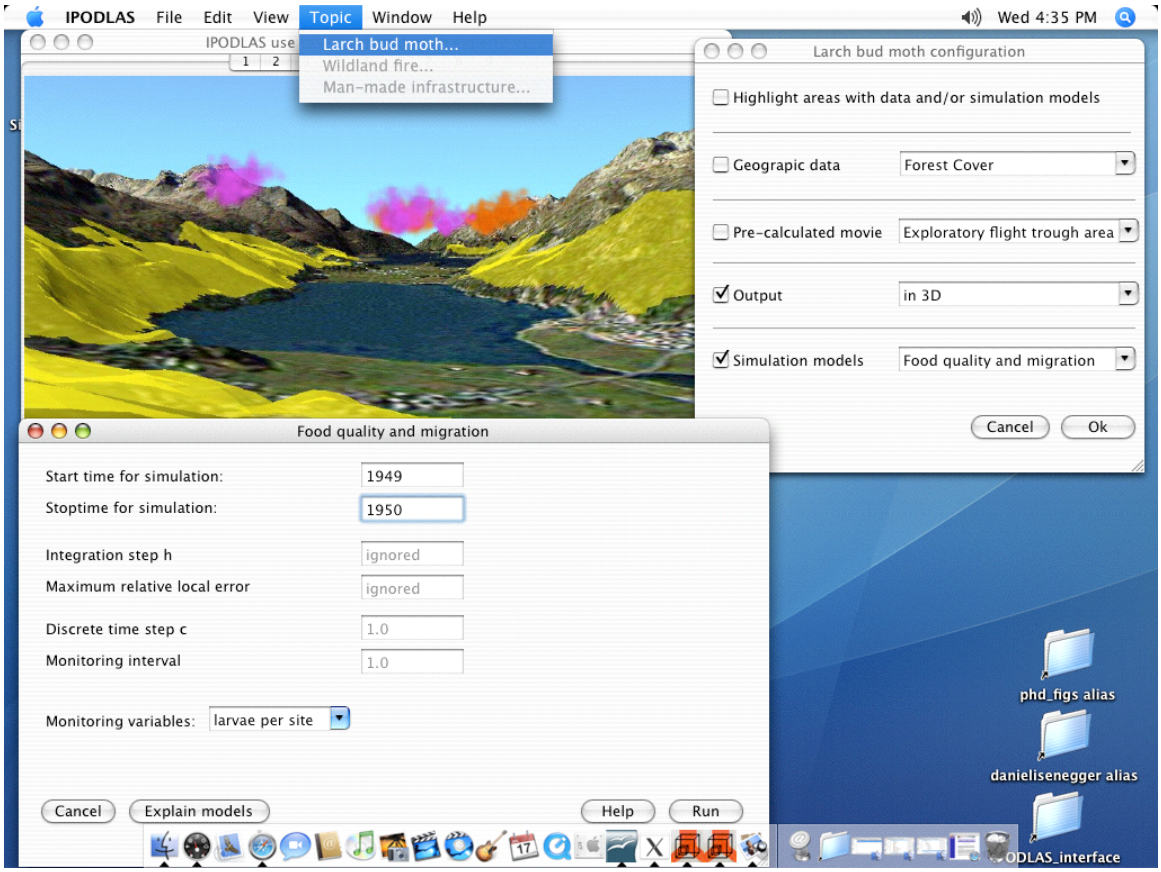

Figure 5-8 A screenshot of the IPODLAS GUI defining graphical elements used in an LBM use case (Isenegger et al., 2005).

5.2.1 Overview of use cases developed within the IPODLAS framework

The use cases specified in the IPODLAS framework are used on the one hand to develop the IPODLAS prototypes, which are described in this thesis (cf. chapter 6 and 7). On the other hand, the use cases are applied in research projects of other members of the IPODLAS project team. For instance, the research scenario described in the use case 'LE2' (cf. Table 5-3) is detailed in subsection 6.1.1 and the use case 'LE4' in subsection 6.1.2. In each case study several use cases have been specified. The specification comprises the prose description, the sequenced action lists, and the associated specification of the GUI. The two types of users — the pilot user and the expert user — are applied in the use cases. Table 5-3 lists all use cases developped in the IPODLAS framework, which form the use case model.

Case study	Use case	Description
LBM	LE	The expert user wants to simulate LBM dynamics and defoliation in the Upper Engadine under the assumption of the food quality hypothesis of the last 50 years and for the next 50 years in the future. She/he wants to see a 3-D visualization of LBM dynamics and to save the simulation results (including the resultant 3-D movie).
LBM	LE2	The expert user is interested particularly in the migration of the LBM across the Upper Engadine valley. She/he wants to see how far LBM migrate per season taking into account wind speed and direction and elements of the landscape which may affect LBM flight such as slope, aspect, and local temperature. The user wants to see the output in 2-D and 3-D and to save the results.
LBM	LE3	The expert user wants to see results of a LBM simulation modeling LBM migration within the Upper Engadine valley. She/he is in particular interested in a tabular representation of migration patterns and a 3-D visualization of the seasonal LBM migration from given sites to other sites and the resultant forest appearance due to defoliation.
LBM	LE3 ext	Extension to LE3: The expert user changes parameters of the LBM simulation model, in particular she/he simulates warmer winter conditions with higher winter egg mortality. Additionally, the user is interested in the interaction of LBM and WLF. She/he wants to explore WLF spread in a LBM peak year exhibiting high defoliation and tree mortality.
LBM	LE4	The expert user wants to see LBM migration within the Upper Engadine valley, in particular a 3-D visualization of the seasonal LBM migration from given sites to other sites and the resultant forest appearance due to defoliation. In addition to LE3, in LE4 the user applies data with higher resolution to investigate how important data resolution is for modeling LBM dynamics.
WLF	WP	The pilot user wants to simulate fire spread in the SNP. She/he chooses 3-D simulation and configures his/her view of the area. The user chooses a default WLF simulation model proposed by the IPODLAS system and configures the simulation by setting the simulation time and ignition point. At the end of the simulation, she/he saves all the results and the simulation as a movie.
WLF	WP ext	Extension to WP: the pilot user stops the WLF simulation and increases the wind speed parameter. Then she/he restarts the simulation again and saves all the results and the simulation as a movie at the end.
WLF	WP2	The pilot user wants to simulate fire spread in the SNP. She/he chooses 3-D simulation and configures his/her view of the area. Then she/he chooses a default WLF simulation model proposed by the IPODLAS system and configures the simulation by setting the simulation time and ignition point. She/he executes several simulation runs with different simulation parameters (e.g. different fire origin, different simulation times, etc) and explores the animations of the WLF simulation by looking at them in the IPODLAS framework from different observer positions. At the end of the simulation, she/he saves all the results and simulation(s) as movie(s).
WFV	VP	The pilot user wants to run a WLF simulation in the SNP and observe the WLF spread from different viewing positions and distances. She/he starts a WLF applying the default WLF simulation model and looks at the fire from a close-by position. Then, while continuously observing the fire, the user increases the observation.

Table 5-3 The use case model consists of all use cases developed in the IPODLAS framework. The names of the use cases in the column 'Use Case' are abbreviations applying the following naming scheme: 'L' stands for LBM, 'W'' for WLF, and 'V' for WFV. Then 'E' is for expert and 'P' for pilot, and the optional number denotes the number of the use case. So, for instance, 'LE2' means LBM expert, use case number two.

In subsections 5.2.2 and 5.2.3, the further development of the above use cases is described in an exemplary way by applying the representative use cases 'LE2' and 'LE3 ext', respectively. The implementation of these use cases is described in the final IPODLAS system described in chapter 7.

5.2.2 Use case 'LBM expert 2' (LE2)[7]

Table 5-4 characterises the user of the use case 'LE2' and her general research intentions. The prose form of the use case 'LE2' (cf. Table 5-5) describes the scenario of a usage of the IPODLAS system by the expert user Bronwyn (Price et al., 2005). In the sequenced action list (cf. Table 5-6) the prose form of the use case 'LE2' is detailed in the sequence of interactions of the user with the system (Price et al., 2005). The sequenced action list details the use case 'LE3 ext' by specifying the interactions of the user with the system and its response.

Bronwyn is an expert user of the IPODLAS system. She is a PhD student within the IPODLAS project and wants to use functionalities of the IPODLAS system to help her answer research questions concerning the spatiotemporal LBM dynamics at differing scales and to investigate the influence of spatial data resolution on modeling LBM dynamics in the Upper Engadine Valley.

Table 5-4 Use case 'LE2' — Description of the user Bronwyn (Isenegger et al., 2005).

Bronwyn is interested particularly in the migration of LBM across the Upper Engadine valley. She wants to see how far LBM migrate per season taking into account wind speed and direction and elements of the landscape which may affect LBM flight such as slope, aspect, and local temperature.

Table 5-5 Use case 'LE2' — Description of the use case 'LE2' in prose form (Isenegger et al., 2005).

Action	Description of action
LE2-1	Bronwyn starts the IPODLAS system and selects LBM from the list of topics.
LE2-2	The IPODLAS GUI shows her the Alpine Arc with highlighted areas where LBM data are provided. Bronwyn selects the Upper Engadine valley.
LE2-3	The IPODLAS GUI displays a 2-D map of the Upper Engadine valley. An additional menu shows several options (geographic data, 3-D, simulate, pre-calculated movie). Bronwyn chooses to simulate and see the output in 2-D.
LE2-4	Bronwyn chooses a start and stop time (1951, 1952) and otherwise keeps all defaults, then runs the model.
LE2-5	The IPODLAS GUI displays a 2-D visualization of the output showing comparative numbers of LBM migrating (departure and landing points).

Table 5-6 Use Case 'LE2' — Sequenced action list (Isenegger et al., 2005).

[7] This section is based on section 4.4 of Isenegger et al (2005).

5.2.3 Use case 'LBM expert 3 extended' (LE3 ext)

The user Bronwyn is the same as in section 5.2.2. The prose form of the use case 'LE3 ext' (cf. Table 5-7) describes the goals of the expert user Bronwyn (Price et al., 2005). Analogous to subsection 5.2.2, the prose form of the use case 'LE3 ext' is specified in greater detail in the sequenced action list (cf. Table 5-8).

The expert user Bronwyn wants to see LBM migration within the Upper Engadine valley, in particular a 3-D visualization of the seasonal LBM migration from given sites to other sites and the resultant forest appearance due to defoliation. After a simulation run, Bronwyn changes parameters of the LBM simulation model in the IPODLAS GUI: she wants to simulate warmer winter conditions with higher egg mortality. Afterwards, Bronwyn is interested in the interaction of LBM and WLF. She wants to explore WLF spread in a LBM peak year exhibiting high defoliation and tree mortality. Bronwyn starts a WLF in a site having a high defoliation rate.

Table 5-7 Use case 'LE3 ext' — Description of the use case 'LE3 ext' in the prose form.

Action	Description of action
LE3e-1	Bronwyn starts the IPODLAS system and selects LBM from the list of topics.
LE3e-2	The IPODLAS GUI shows her the Alpine Arc with highlighted areas where LBM data are provided. Bronwyn selects the Upper Engadine valley.
LE3e-3	The IPODLAS GUI displays a 2-D map of the Upper Engadine valley. An additional menu shows several options (geographic data, 3-D, simulate, pre-calculated movie). Bronwyn chooses to simulate and see the output in 3-D.
LE3e-4	In the LBM configuration window, Bronwyn chooses a start and stop time (1953, 1954) and otherwise keeps all defaults, then runs the model.
LE3e-5	The IPODLAS GUI displays a tabular output of the simulation results and a 3-D visualization of the output showing comparative numbers of LBM migrating (departure and landing points) and the resultant coloring of the forest in the sites symbolizing the defoliation ratio.
LE3e-6	After the end of the simulation, Bronwyn changes the 'winter egg mortality' parameter in the LBM configuration window, keeps all the other defaults, and starts the simulation again.
LE3e-7	The IPODLAS GUI displays a 3-D visualization of the output showing comparative numbers of LBM migrating (departure and landing points) and the resultant coloring of the forest in the sites symbolizing the defoliation ratio.
LE3e-8	After the simulation ended, Bronwyn selects in the IPODLAS GUI the WLF topic and then the 'WLF simulation configuration' window. In there, Bronwyn chooses that the ignition point can be set in VR and that the 'live moisture' parameter of the WLF simulation model is taken from the output of the LBM simulation, and then starts the simulation.
LE3e-9	In the VR GUI the defoliation of the forest in the respective sites is displayed applying different colors. Bronwyn now selects a WLF ignition point in a site exhibiting high defoliation values.
LE3e-10	In the VR GUI the spread of a WLF is shown starting from the user-selected ignition point.

Table 5-8 Use Case 'LE3 ext' — Sequenced action list.

5.3 Required subsystem functionality

5.3.1 The functionality listing[8]

The use cases require functionality provided from several subsystems. The subsections 5.2.2 respectivley 5.2.3 describe the IPODLAS system executing the use cases 'LE2' respectively 'LE3 ext'. The use cases and sequenced action lists specify the users view of the IPODLAS system. Since they interact only with the IPODLAS GUI, the subsystems are hidden; the users see the IPODLAS system as one monolithic system.

The functionality listing investigate the mode of operation of the IPODLAS system on the level of the subsystems. Applying the use cases 'LE2' respectively 'LE3 ext' the functionality listing in Table 5-9 lists the interactions between the subsystems of the IPODLAS system. The actions of the two use cases in Table 5-9 are listed according to their temporal occurence. To be able to to specify the interactions of the subsytems the actions specified the sequenced action lists Table 5-7 respectively Table 5-8 are described in substeps.

Action	Sub-system	Required functionality	Type of functionality
LE2-1, LE3e-1	IPODLAS GUI	Displaying widgets for user interaction: for selecting topic, area, etc.	Display widgets and catching user input
LE2-2a, LE3e-2a	IPODLAS GUI	Requesting spatial data from the GIS and non-spatial data from the IPODLAS storage	Communication / information exchange between subsystems
LE2-2b, LE3e-2b	GIS	Receiving a request from the IPODLAS GUI to provide areas with available data	Communication / information exchange between subsystems
LE2-2c, LE3e-2c	GIS	Retrieving requested spatial data	Spatial query
LE2-2d, LE3e-2d	GIS	Providing the IPODLAS GUI the requested spatial data	Communication / information exchange between subsystems
LE2-2e, LE3e-2e	IPODLAS storage	Receiving from the IPODLAS GUI a request to provide non-spatial data	Communication / information exchange between subsystems
LE2-2f, LE3e-2f	IPODLAS storage	Querying requested non-spatial data	Query
LE2-2g, LE3e-2g	IPODLAS storage	Providing the IPODLAS GUI the requested non-spatial data	Communication / information exchange between subsystems
LE2-2h, LE3e-2h	IPODLAS GUI	Receiving requested data from the GIS and the IPODLAS storage	Communication / information exchange between subsystems
LE2-3, LE3e-3	IPODLAS GUI	Displaying widgets for user interaction: for configuring LBM simulation	Display widgets and catching user input

[8] This subsection is based on section 4.4 of Isenegger et al (2005).

LE2-4a, LE3e-4a, LE3e-6a	IPODLAS GUI	Requesting: - TSS to execute a LBM simulation with user-defined simulation parameter values - VR to load required data for visualization of research area	Communication / information exchange between subsystems Communication / information exchange between subsystems
LE2-4b, LE3e-4b, LE3e-6b	TSS	Receiving LBM simulation request from IPODLAS GUI	Communication / information exchange between subsystems
LE2-4c, LE3e-4c, LE3e-6c	TSS	Requesting spatial data from the GIS and non-spatial data from the IPODLAS storage	Communication / information exchange between subsystems
LE2-4d, LE3e-4d, LE3e-6d	GIS	Receiving request concerning spatial data from TSS (forest and in particular larch distribution, calculating slope and aspect, and wind simulation)	Communication / information exchange between subsystems
LE2-4e, LE3e-4e, LE3e-6e	GIS	Requesting non-spatial data from the IPODLAS storage	Communication / information exchange between subsystems
LE2-4f, LE3e-4f, LE3e-6f	IPODLAS storage	Receiving request from the GIS to provide non-spatial data	Communication / information exchange between subsystems
LE2-4g, LE3e-4g, LE3e-6g	IPODLAS storage	Querying requested data	Query
LE2-4h, LE3e-4h, LE3e-6h	IPODLAS storage	Providing the requested non-spatial data to the GIS	Communication / information exchange between subsystems
LE2-4i, LE3e-4i, LE3e-6i	GIS	Receiving requested non-spatial information from IPODLAS storage	Communication / information exchange between subsystems
LE2-4j, LE3e-4j, LE3e-6j	GIS	Calculating: - larch per hectare, forest area per hectare, temperature distribution - slope, aspect - wind speed and direction statistics	Map algebra, map overlay, clipping Slope, aspect calculation Simulation using the wind model and calculating statistics using Map Algebra
LE2-4k, LE3e-4k, LE3e-6k	GIS	Providing the TSS the requested spatial data	Communication / information exchange between subsystems
LE2-4, LE3e-4l, LE3e-6l	TSS	Receiving requested information from GUI	Communication / information exchange between subsystems

LE2-4m, LE3e-4m, LE3e-6m	TSS	Executing the user-selected LBM simulation model	Simulation with user-selected LBM simulation model and parameter values
LE2-4n, LE3e-4n, LE3e-6n	TSS	Providing the requested simulation results to the IPODLAS GUI and requesting to write LBM simulation output to the IPODLAS storage	Communication / information exchange between subsystems
LE2-4o, LE3e-4o, LE3e-6o	IPODLAS storage	Receiving write request from the TSS	Communication / information exchange between subsystems
LE2-4p, LE3e-4p, LE3e-6p	IPODLAS storage	Writing LBM simulation output to the IPODLAS storage	Write data to IPODLAS storage
LE2-4n, LE3e-4n, LE3e-6n	VR	Requesting spatial data from the GIS for visualizing	Communication / information exchange between subsystems
LE2-4o, LE3e-4o, LE3e-6o	GIS	Receiving spatial data request from VR	Communication / information exchange between subsystems
LE2-4p, LE3e-4p, LE3e-6p	GIS	Retrieving requested spatial data	Spatial query
LE2-4q, LE3e-4q, LE3e-6q	GIS	Providing the requested spatial data to the VR	Communication / information exchange between subsystems
LE2-4r, LE3e-4r, LE3e-6r	VR	Receiving requested spatial data from the GIS	Communication / information exchange between subsystems
LE2-5a, LE3e-5a, LE3e-7a	IPODLAS GUI	Receiving tabular LBM simulation output from the TSS	Communication / information exchange between subsystems
LE2-5b, LE3e-5b, LE3e-7b	IPODLAS GUI	Displaying tabular LBM simulation output and requesting the VR to visualize LBM simulation output	Communication / information exchange between subsystems
LE2-5c, LE3e-5c, LE3e-7c	VR	Receiving request to visualize LBM simulation output from the IPODLAS GUI	Communication / information exchange between subsystems

LE2-5d, LE3e-5d, LE3e-7d	VR	Visualizing LBM simulation output and spatial data	Visualizing tabular and raster data
LE2-5e, LE3e-5e, LE3e-7e	VR	Notifying the GUI when visualization has finished	Communication / information exchange between subsystems
LE2-5f, LE3e-5f, LE3e-7f	IPODLAS GUI	Receiving notification and informing user via displaying widgets	Communication / information exchange between subsystems
LE3e-8a	IPODLAS GUI	Displaying widgets for user interaction: for configuring the WLF simulation	Display widgets and catching user input
LE3e-8b	IPODLAS GUI	Requesting: - VR to visualize WLF in defoliated sites - VR to load requested data for visualization - WLF simulation with user-defined simulation parameter values from the GIS	Communication / information exchange between subsystems Communication / information exchange between subsystems
LE3e-8c	VR	Receiving request from IPODLAS GUI concerning visualization of a WLF of defoliation data in sites	Communication / information exchange between subsystems
LE3e-8d	VR	Requesting defoliation data in sites from GIS	Communication / information exchange between subsystems
LE3e-8e	GIS	Receiving spatial data request from the VR	Communication / information exchange between subsystems
LE3e-8f	GIS	Retrieving requested spatial information	Spatial query
LE3e-8g	GIS	Providing the requested spatial data to the VR	Communication / information exchange between subsystems
LE3e-8h	VR	Receiving requested defoliation information from the GIS	Communication / information exchange between subsystems
LE3e-8i	VR	Visualizing defoliation data	Visualizing tabular data
LE3e-9a	VR	Catching WLF ignition point selected by the user on the VR GUI	Reading the coordinates the pointing device points to in VR
LE3e-9b	VR	Requesting WLF simulation (with ignition point) from the GIS	Communication / information exchange between subsystems
LE3e-9c	GIS	Receiving request concerning WLF simulation (with ignition point) from the VR	Communication / information exchange between subsystems
LE3e-9d	GIS	Requesting defoliation values of sites from IPODLAS storage	Communication / information exchange between subsystems
LE3e-9e	IPODLAS storage	Receiving request to provide non-spatial data from the GIS	Communication / information exchange between subsystems
LE3e-9f	IPODL	Querying requested data	Query

	AS storage		
LE3e-9g	IPODL AS storage	Providing the requested non-spatial data to the GIS	Communication / information exchange between subsystems
LE3e-9h	GIS	Receiving requested defoliation data from the IPODLAS storage	Communication / information exchange between subsystems
LE3e-9i	GIS	Executing the WLF simulation	Accessing and invoking a WLF simulation model to simulate a WLF spread
LE3e-9j	GIS	Providing the requested WLF simulation to the VR	Communication / information exchange between subsystems
LE3e-9k	VR	Receiving the requested WLF simulation from the GIS	Communication / information exchange between subsystems
LE3e-9l	VR	Visualizing the WLF simulation output	Visualizing tabular and raster data

Table 5-9 The functionality listing of the use cases 'LE2' respectively 'LE3 ext', based on a initial compilation of Isenegger et al. (2005). The listings lists functionality required by the IPODLAS system from the respective subsystem respectively the legacy systems. The column 'Action' exhibits the number of the action in the use cases 'LE2' and 'LE3 ext'.

5.3.2 Analysis of the functionality listing[9]

The Table 5-9 gives an idea of the *division of labour* on the subsystem level: it shows which subsystem is requested to execute when which part of the requests required in the use case 'LE2' respectively 'LE3 ext'. In general, the actions of the Table 5-9 are ordered according to the time they occur, but the order of the actions is not strictly temporally. On the one hand, although one action is listed below an other action in Table 5-9 the two actions (of two subsystems) may happen concurrently. On the other hand to avoid repetitions the actions LE3e-6a to LE3e-6o are listed in the same lines as LE2-4a to LE2-4o respectively LE3e-4a to LE3e-4o due to their similar sequence of actions. In section 7.3 the implementation of the use case 'LE3 ext' is detailed like in Table 5-9, but applying a finer grain listing the actions "taking place behind the scene" between the subsystems.

For the specification of the sequence of actions listed in Table 5-9 it has been assumed that the IPODLAS system comprises a *common IPODLAS GUI* and a *common IPODLAS storage*. The user communicates with the entire IPODLAS system via the common IPODLAS GUI; the only exception is that the user selects the ignition point of the WLF spread on the VR GUI, where the defoliation is visualized. The common IPODLAS storage is a persistent storage, which is accessible by all subsystems of the IPODLAS system. It holds all non-spatial data, which is may of interest for the subsystems of the IPODLAS system, for example results of previous simulation runs or pre-calculated data sets. For the storage of spatial data the GIS offers the second persistent storage within the IPODLAS system.

Classification of the required functionality

The domain-typical and communication functionality can be classified using the classifiction defined in Table 5-2. All domain-typical functionality required in the use cases 'LE2' respectively 'LE3 ext' from the subsystems by the IPODLAS system can be

[9] This subsection is based on section 4.4 of Isenegger et al (2005).

provided the respective legacy systems. Thus, the domain-typical functionality can be classified into class 1 of the classification specified in Table 5-2. The communication functionality can be classified into functionality classes 2, 3, or 4 of the classification specified in Table 5-2 depending on the conceptual and technical complexity of the chosen solution to provide this functionality. As an example, in the action sequence LE2-4i to LE2-4k the classification of this task into class 2 could mean that data is sent only as simple text file to the requesting subsystem, while class 3 indicates a more advanced solution such as the automatic encoding of spatiotemporal data in GML 3 for sending data (this will be explained in subsection 6.2.1).

Communication and domain-typical functionality

The functionality listing in Table 5-9 is typical for the functionality listings of use cases developed for the IPODLAS system. The functionality specified in the listing can be divided roughly into two types. The first type of functionality required in the use cases 'LE2' respectivley "LE3 ext' is *communication functionality*; in Table 5-9 this type of functionality is named 'Communication / information exchange between subsystems'. This functionality is used to synchronized the individual subsystems and to exchange information between the subsystems. The second type of functionality is *domain-typical functionality* of the legacy systems of the subsystems TSS, VR, and GIS. Temporal simulation functionality is domain-typical functionality of legacy systems of TSS, visualization and user interaction functionality is domain-typical for legacy systems of VR, and functionality for dealing with spatial problem is domain-typical for legacy systems in GIS. Table 5-10 lists the actions of the use cases 'LE2' respectively 'LE3 ext' (numbered according to Table 5-9), where this second type of functionality is applied.

Actions	Subsystem	Type of functionality
LE2-4m, LE3e-4m, LE3e-6m	TSS	Temporal simulation
LE2-5d, LE3e-5d, LE3e-7d, LE3e-8i, LE3e-9i	VR	Visualization
LE2-2c, LE3e-2c, LE2-4j, LE3e-4j, LE3e-6j, LE2-4p, LE3e-4p, LE3e-6p, LE3e-8f, LE3e-9i	GIS	Spatial and thematic search, locational analysis, terrain analysis, spatial simulation
LE2-1, LE3e-1, LE2-3, LE3e-3, LE3e-8a	IPODLAS GUI	Display widgets and catch user input
LE2-2f, LE3e-2f, LE2-4g, LE3e-4g, LE3e-6g, LE2-4p, LE3e-4p, LE3e-6p, LE3e-9f	IPODLAS storage	Query functionality

Table 5-10 Classification of actions requiring domain-typical functionality of the use cases 'LE2' and 'LE3 ext' (cf. Table 5-9). All actions requiring domain-typical functionality of the two use cases can be classified using the listed types of functionality.

When comparing Table 5-10 with the actions of Table 5-9, it is shown that all actions of the 'domain-specific functionality'-type in Table 5-9 can be classified into the respective classes in Table 5-10. This means that all domain-typical functionality required in the use cases can be provided by the respective legacy systems. In the case of the required spatial functionality all actions requiring spatial functionality listed in Table 5-9 can be classified using the *taxonomy of Albrecht* (1997): No other GIS functionality than *spatial* and *thematic search*, *location analysis*, and *terrain analysis* (cf. Table 5-10, 3rd row) is required in the use cases. Thus, GIS which provide the range of standard functionality identified by Albrecht can provide the spatial functionality required by the IPODLAS system. An advanced analysis of the required spatial functionality by IPODLAS which may discover other

spatial functionality requires the investigation of more use cases of case stories from different domains. Due to the limitation of Albrecht's typology to analytical functionality the *spatial simulation* required by the IPODLAS system (cf. actions LE2-4j, LE3e-4j, LE3e-6j, and LE3e-9i in Table 5-9 and the corresponding 3rd row in Table 5-10) cannot be classified using the taxonomy of Albrecht.

Table 5-9 shows that the communication functionality deals with both the *control and the data flow*. The actions LE2-4n to LE2-4r show a typical communication sequence of subsystems of the IPODLAS system in the use case 'LE2'. In action LE2-4n the VR subsystem requests a service from another subsystem, in this case a request for spatial data from the GIS. The GIS receives the request in action LE2-4o, executes the request in action LE2-4p, and notfies the requesting subsystem in LE2-4q. In action LE2-4r, the VR subsystems receives the requested spatial information. On a conceptual level, in action LE2-4n to LE2-4q the control flow is affected, i.e. for each subsystem seamless access of functionality of each other subsystem is required. Action LE2-4r address the data flow, i.e. for each subsystem seamless access of data of each other subsystem is required.

At least until this stage of the development, this means that all domain-typical functionality required in the use cases can be provided by the applied subsystems respectively actually of the applied legacy systems. The IPODLAS system requires no domain-typical functionality which cannot be provided by the legacy systems. In contrast, the communication functionality required by the IPODLAS system from the subsystems cannot be provided straightforwardly by the respective legacy system. The communication functionality allowing the individual legacy systems to interact must be provided by the IPODLAS system. This means, that when implementing the use cases the challenges that are occurring at this stage of the IPODLAS system development are not missing domain-typical functionality, but rather the communication functionality required for the interaction of the subsystems

6 Bringing TSS, VR, and GIS together

The key concept of IPODLAS project is the combined usage of functionality of the three domains to better represent spatiotemporal, cross-scale natural processes. In section 6.1 research conducted by members of the IPODLAS project is presented. In all projects functionality of more than one domain was required to address the research questions in a satisfying way. Section 6.2 describes the evolution of aspects of the software architecture of the final IPODLAS system detailed in chapter 7: Different preliminary prototypes implementing important aspects of the final IPODLAS system are explained in this section.

6.1 The added value of the combined usage of TSS, VR, and GIS

It is suggested that research questions of the research subprojects conducted within the IPODLAS project can be addressed in a more comprehensive way by applying combined functionality of more than one domain compared to an isolated approach. Use cases developed in the IPODLAS framework (cf. subscetion 5.2.1) are applied in research projects of members of the IPODLAS project team: Subsection 6.1.1 discusses the application of the use case 'LE2' in the research project conducted by Price (2005) whereas subsection 6.1.2 presents the application of the use case 'LE4' in the research project of Price et al. (submitted). The functionality developed and applied in the different use cases have been implemented in several protoypes. The open source GIS GRASS (cf. subsection 3.3.4) was applied to provide spatial functionality described in 6.1.1 and 6.1.2.

The author of this thesis is co-author of the paper 'Spatiotemporal modelling of Larch bud moth in the European Alps: The importance of data resolution' (Price et al., submitted) and of 'The influence of orography on Larch bud moth migration at the valley scale' (Price, 2005). Thus, some paragraphs in this thesis are cited verbatim from Price et al. (submitted) or Price (2005), respectively. This is indicated by a footnote "Cited from Price et al. (submitted) or "Cited from Price (2005)" at the end of the respective paragraph.

6.1.1 LBM-GIS: an LBM migration model

The use case 'LE2' described in subsection 5.2.1 concentrates on the influence of spatial variable data on simulation models. LBM dynamics (cf. subsection 5.1.1) in the Upper Engadine valley show considerable temporal (Fischlin, 1982) and spatial synchrony, that is, LBM populations fluctuate in separate locations within the valley concurrently in a similar manner (Price, 2005). If the migration of the LBMs is a driver for the observed spatial patterns, the characteristics of the landscape and the conditions under which the migration takes place may are of crucial importance. Such influencing factors of the LBM migration can be the migration distance, the topography, and wind conditions (Price, 2005).

Some existing simulation models of LBM dynamics include spatially varying attributes, but only in a restricted manner (Price, 2005). For instance, the LBM dynamics simulation model LBM-M9 (Fischlin, 1982) (cf. subsection 5.1.4) parameterizes spatially distributed attributes such as wind conditions, migration distances, and forested areas applying a coarse spatial resolution. It is suggested that the ability of LBM simulation models to predict actual LBM densities is dependent on the spatial scale of the input data, in particular of the spatial grain. Thus, the use of spatially explicit data on the appropriate spatial scale may contribute to the generation of more accurate LBM

simulation models (Price, 2005). In the use case 'LE2' of the IPODLAS framework the combination of spatial functionality with LBM simulation knowledge derived from LBM-M9 is applied in the LBM migration model called *LBM-GIS*. The effect of the LBM-GIS model on the accuracy of LBM dynamics models is described in greater detail in Price (2005). This subsection illustrates how the LBM migration modeled in LBM-M9 can be improved by the application of spatial (GIS) functionality.

In the LBM-M9 model the LBM migration is modeled as migration originating from each site center applying wind statistics[1] and ignoring topography. The wind statistics describe the frequencies of wind in one of sixteen compass directions (NNW, NW, WNW, ..., ENE, NE, NNE) and the respective wind speeds. In the GIS-empowered LBM-GIS model in contrast the migration of the LBMs is assumed to be influenced by the three factors *favored flight direction*, *maximal potential flight distance* and *topography*. LBMs show different favored flight directions: One fraction of all LBMs only flies uphill or across flat areas[2] against the wind (*upwind flyers*), whereas another fraction only flies downhill or across flat areas with the wind (*downwind flyers*). Due to conceptual similarity of the implementation of upwind and downwind flyers in the LBM-GIS, in the following example only upwind flyers are modeled. The modeled favored flight direction is one of the sixteen compass directions and is taken as well as the maximal potential flight distance from LBM-M9 (Price, 2005). The influence of topography is twofold: (Upwind) LBMs are assumed firstly not to fly higher than 2100 m.a.s.l. and secondly only to fly across flat areas or uphill (these are denoted as positive slopes).

The approach applied in the LBM-GIS model to describe LBM migration is to model the migration of the LBMs considering the costs of the different factors. The LBM-GIS model applies GRASS GIS (cf. 3.3.4) to generate for each of the factors elevation, slope, and wind a raster whose cell values represent costs for a LBM to cross the respective cell (cf. Figure 6-1):

1. Elevation raster: In all cells with elevation value higher than 2100 m.a.s.l no migration is assumed, i.e. the cells obtain prohibitively high migration costs.
2. Slope raster: The slope raster shows low migration costs in cells with positive slope with respect to the LBM flight direction.
3. Wind raster: LBMs are "able to fly in the 22.5° sector of their favored direction [...] to a maximum distance calculated by the LBM-M9 from the center of each site" (Price, 2005, p. 67). So, all raster cells located within this sector exhibit small migration cost. Cells located in adjacent sectors show higher migration cost, while in all other raster cells no migration is assumed. The maximal potential flight distance is then superimposed and limits thus the potential flight areas.

r.mapcalc[3], the GRASS implementation of Map Algebra (Tomlin, 1990), has been applied to generate a joint migration cost raster data set by summing the three rasters. This joint migration cost raster is used to calculate the cost surface by applying the GRASS command *r.cost*[4], which computes the cumulative costs of moving on the joint migration cost raster. The cost surface exhibits in each cell the cumulative cost to migrate from given starting cells to the particular cell (Shapiro, 1991); in Figure 6-1 the cost surface calculated from the three (cost) rasters 'Elevation', 'Slope', and 'Wind and maximal

[1] "Wind speed and direction for each site was derived from measurements taken at the Swiss Federal Office of Meteorology and Climatology weather stations […]" (Price, 2005, p. 67). The Swiss Federal Office of Meteorology and Climatology (MeteoSwiss) can be accessed under http://www.meteoswiss.ch/web/en.html (accessed February 20, 2006).

[2] Price (2005, p. 67) states "we define here 'flat' areas as those with slopes between 0° and 5°".

[3] http://grass.itc.it/grass60/manuals/html60_user/r.mapcalc.html (accessed February 21, 2006).

[4] http://grass.itc.it/grass60/manuals/html60_user/r.cost.html (accessed February 20, 2006).

distance' is labeled 'Potential flight areas' and shows possible flight areas of the LBMs allowing to determine to which sites LBMs can migrate.

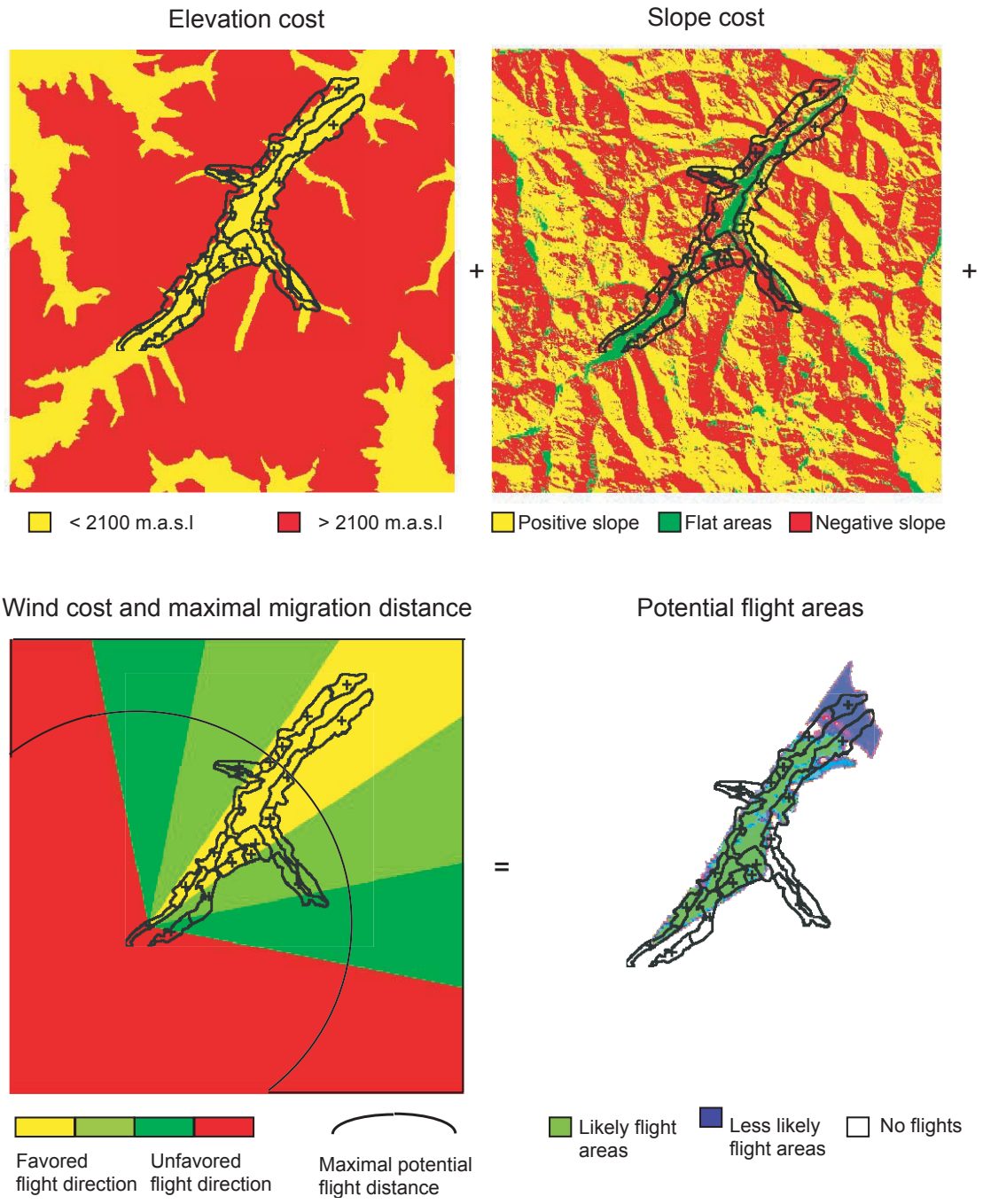

Figure 6-1 Conceptual illustration of the generation of the potential flight areas of LBMs; in the example the upwind flyers fraction. The example shows the calculation of migration from site 1 — the most western site — with wind direction NE (the shape of the sites are depicted in black in each raster). The three rasters 'Elevation cost', 'Slope cost', and 'Wind cost and maximal migration distance' are used to calculate the 'Potential flight areas', which shows the likelihood that LBMs from site 1 and main wind direction NE migrate to the respective areas.

6.1.2 The cross-scale approach

The use case 'LE4' (cf. subsection 5.2.1) described in this subsection addresses the application of simulation models on different spatial scales. In Price et al. (submitted) a major research question was, if there is an *optimal spatial scale*[5] for the application of the LBM simulation models to get the most accurate model results. In particular, the influence of increased spatial resolution on the model results was analyzed. The LBM case study of the IPOLDAS case study framework provides three LBM simulation models defined on the same spatial extent (the Upper Engadine Valley) but applying different spatial resolutions (Price et al., submitted). The LBM-M8 (Fischlin, 1982) models LBM dynamics perceiving the whole Upper Engadine as one point in space applying the *valley* scale. The LBM-M9 (Fischlin, 1982) considers the *site* as smallest spatial grain, and the LBM-M11 (Price, 2005) is defined on the *forest compartment* level (cf. subsection 5.1.1 for the levels of spatial resolution and subsection 5.1.4 for the LBM simulation models). Price et al. (submitted) combine LBM-M11 with the GRASS GIS (Neteler and Mitasova, 2002) to employ GIS functionality for acquiring spatial information. Since in LBM-M11 the same LBM simulation functionality is used as in LBM-M9, both simulation models require the same spatial data to simulate LBM dynamics, each at the appropriate scale. Table 6-1 shows the respective spatial data input for the different simulation model parameters of LBM-M9 and LBM-M11.

Model parameter	**Data source (LBM-M9)**	**Data source (LBM-M11)**
Frequency of turbulence in research area *i*	MeteoSwiss, 1901-1990 (Fischlin, 1982)	MeteoSwiss, 1901-1990
Frequency of calm winds (0-0.5m/s) in research area *i*	MeteoSwiss, 1901-1990	Simulated in NUATMOS (Ross et al., 1988) and calculated in GRASS (Neteler and Mitasova, 2002)
Frequency of calm winds (0.5-2.8m/s) in research area *i*	MeteoSwiss, 1901-1990	Simulated in NUATMOS and calculated in GRASS
Frequency of strong winds (> 2.8 m/s) in research area *i*	MeteoSwiss, 1901-1990	Simulated in NUATMOS and calculated in GRASS
Frequency of calm winds in research area *i* in direction *j*	MeteoSwiss, 1901-1990	Simulated in NUATMOS and calculated in GRASS
Frequency of strong winds in research area *i* in direction *j*	MeteoSwiss, 1901-1990	Simulated in NUATMOS and calculated in GRASS
Area of neighboring research area *n* in direction *j* in sub-sector A resp. B	MeteoSwiss, 1901-1990	n/a
Area of neighboring research area *n* in sector *j*	n/a	Calculated in GRASS
Air distance from research area *i* to neighboring research area *n* in sub-direction A resp. B	MeteoSwiss, 1901-1990	n/a
Air distance from research area *i* to neighboring research area *n* in direction *j*	n/a	Calculated in GRASS

Table 6-1 Spatial data required as input to model LBM-M9 and LBM-M11 with sources, after Price et al. (submitted). The research area is represented by the *sites* for LBM-M9 respectively by the *forest compartments* for LBM-M11. The indices range in LBM-M9 is : *i* = 1-20, *n* = 1 – 20, *j* = NE, E, SE, S, SW, W, NW, N . In lBM-M11 the indices range is LBM-M11 *i* = 1-420, *n* = 1-420, *j* = NE, E, SE, S, SW, W, NW, N.

[5] Spatial scale was defined in chapter 1 as magnitude of the area under consideration (spatial extent) and also to the degree of detail (spatial resolution or spatial grain) (Quattrochi and Goodchild, 1997).

The model parameters are described in the first column of Table 6-1; they can be divided roughly into wind statistics and neighborhood statistics. Fischlin (1982) applied parameterized spatial data at site level in LBM-M9. LBM-M11 uses wind field simulation model NUATMOS (Ross et al., 1988) to simulate wind fields and GRASS to calculate wind statistics on the forest compartment level based on the wind fields and neighborhood statistics (cf. third column of Table 6-1). The generation of the wind statistics and the neighborhood statistics used in LBM-M11 is described in the following subsections 'Wind statistics' and 'Neighborhod statistics'.

Wind statistics

It is suggested that due to the highly dynamic nature of atmospheric conditions, the LBM simulation models requiring wind data can considerably benefit from the application of a spatially distributed wind simulation model instead of using static statistical wind data (Price et al., submitted). Additionally, wind observation data at the required spatial resolution (the forest compartment) is not available for the Upper Engadine. Therefore the meso-scale wind simulation model *NUATMOS* (Ross et al., 1988) was applied to generate the wind data required by LBM-M11.

"The wind observation data used as input to NUATMOS was retrieved from the Swiss Federal Office of Meteorology and Climatology (MeteoSwiss)[6]. The six meteorological observation stations chosen to provide initial conditions were within and around the Upper Engadine valley (Bever, Bivio, Corvatsch, Robbia/Poschiavo, Sils Maria, and St.Moritz). As topographical effects largely drive wind patterns within the Upper Engadine valley, average summer wind speed and direction values are considered constant through time by the LBM-M9 and LBM-M11 models (Fischlin, 1982). Therefore, a period for which all of the relevant meteorological observation stations provide data, 1980 to 1982, was chosen from which to take data to drive NUATMOS. Since NUATMOS requires observations from at least one observation station not located on the surface, wind observations from the troposphere (ca. 5500 m.a.s.l.) were also taken".[7]

The wind observation data from MeteoSwiss was fed into a PostgreSQL database (cf. subsection 3.3.4). The required data was queried using SQL in Python scripts and written into 200 input files for NUATMOS.

Based on these wind observations, NUATMOS interpolated wind direction fields and wind speed fields for 200 points in time. NUATMOS produces a "three dimensional mass-consistent windfield based on observations which are arbitrarily located. This is achieved by interpolating [...] and then making minimal adjustments in order to eliminate divergence" (Ross et al., 1988, p. 15). "The input consists of parameters controlling NUATMOS, specification of the digital elevation model (DEM) on which NUATMOS is applied, and wind observations in the form of horizontal wind components. The wind direction and speed on the surface is calculated from the three dimensional wind field NUATMOS generates (Bachmann, 1998). In this study NUATMOS version 5N (07/31/91) (Ross et al., 1988) has been applied to a DEM with a spatial resolution of 50

[6] The Swiss Federal Office of Meteorology and Climatology (MeteoSwiss) can be accessed under http://www.meteoswiss.ch/web/en.html (accessed February 20, 2006). Some wind observation data can be retrieved on this web page. However, for comprehensive wind data retrieval a contract with MeteoSwiss is required.

[7] Cited from Price et al. (submitted, p. 10). The surface wind data has been retrieved from the 'Klima Datenbank' of MeteoSwiss. Since no particular wind observation data from the troposphere for the Upper Engadine exists, wind data of 'Alpenwetterstatistik Datenbank' covering the Swiss alpine region was taken.

m (DHM50 ©, Tydac AG). The accuracy and the usefulness of the wind fields[8] generated by NUATMOS were tested through evaluation of the generated wind fields against values from the meteorological observation station of Samedan. The difference in average wind direction was 62.9 degrees and the observed average wind speed is 3.79 m/s compared to simulated average wind speed of 1.63 m/s. An evaluation of NUATMOS by Connell (1989) has shown that the best agreement between modeled and observed values is achieved at mountain tops whereas poor agreement occurs at low wind speeds (i.e. 2m/s) and when re-circulating flow on the lee side of mountains occurs."[9]

The 3-D wind fields generated by NUATMOS were applied to derive 2-D wind fields, which represent the wind direction and wind speed on the surface of the applied DEM, i.e. of the Upper Engadine valley. The 2-D wind fields were imported into GRASS, where r.mapcalc was applied to calculate the wind statistics (cf. Table 6-1) for each cell. Figure 6-2 shows the rasters exhibiting frequencies of calm, weak and strong winds (cf. Table 6-1). While the generation of statistical values for wind speed requires only (normal) descriptive statistics, the calculation of statistical values for wind direction demands the application of directional statistics due to the modal nature of directions. A crucial concept of directional statistics is to regard a modal value (e.g. a direction) as unit vectors on a circle in the plane and calculate with the polar coordinates (Mardia and Jupp, 2000).

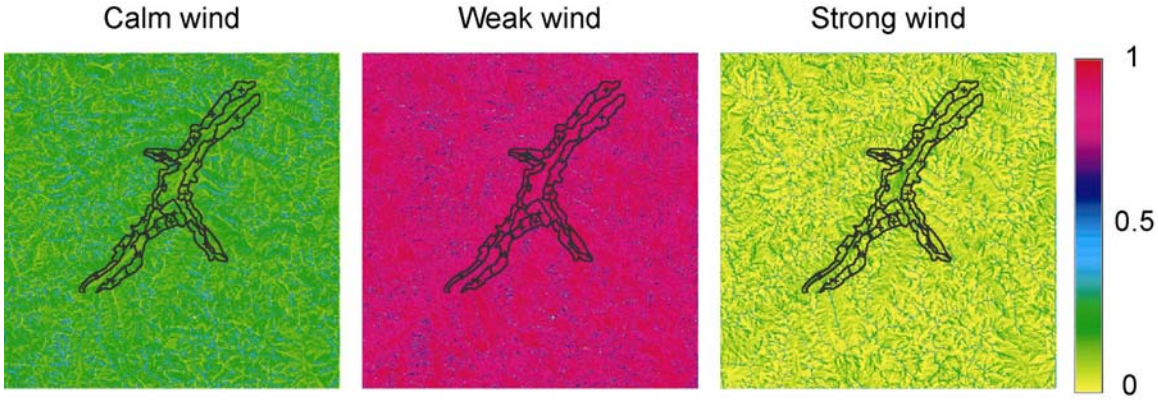

Figure 6-2 Frequency of wind classes between 0 and 1 (cf. Table 6 1, calm winds: 0 - 0.5 m/s, weak wind: 0.5 – 2.8 m/s).

Neighborhood statistics

"To obtain the neighbourhod data for each forest compartment required as input to the LBM-M11 model, the [perimeters of the]forest compartments were first digitized and stored as a vector data layer within a GIS. The model requires knowledge of the nearest neighbors for each forest compartment in each of the eight compass directions. Therefore, forest compartments that are spatially separated and not sharing any borders are still neighbours as long as no other forest compartments are located between them."[10]

The calculation of the neighborhood statistics in LBM-M11 requires the determination of the nearest neighboring forest compartment for all forest compartments in all eight compass sectors (cf. Table 6-1). The most simple solution was to define the distance between the centers of the polygons representing the forest

[8] The accuracy evaluation was conducted with 2-D wind fields, not with the original 3-D wind fields generated by NUATMOS.
[9] Cited from Price et al. (submitted, p. 10).
[10] Cited from Price et al. (submitted, p. 10).

compartments as the distance between the forest compartments. More sophisticated solutions for the definition of the distance between two polygons, as for example the determination of the smallest distance between any two points located on the respective borderlines of the polygons requires much more computation time. Extracting the coordinates of the polygon centers referenced in the cartesian Swiss reference system CH1903[11] from the GIS-DB, the distance and the angle from East between any two polygon centers can be calculated (cf. Figure 6-3).

Thus, it "was possible to determine (i) in which direction each neighbour is located, (ii) the area of each neighbor in the given compass sector and (iii) the distance to each neighbor centre".[12]

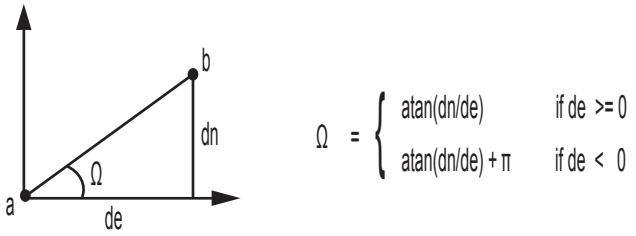

Figure 6-3 Calculation of the angle Ω of point b to the abscissa in respect to point a. When using a Cartesian coordinate system, de respectively dn can be determined directly by subtraction of the abscissas respectively the ordinates.

6.2 Iterative development of the software architecture

To illustrate the iterative approach of the IPODLAS system development, this section outlines some prototypes developed within the IPODLAS framework which delineate findings for the evolution of important features of the IPODLAS system[13]. In subsection 6.2.1 the potentials of GML 3 to represent spatiotemporal data is shown. The 'Intelligent Tree' described in subsection 6.2.2, presents a simple interaction model of the subsystems, while the prototype 'Cross-implementation' detailed in subsection 6.2.3 describes a combination of TSS and GIS denoted as 'embedded' in subsection 3.4.1. Subsection 6.2.4 discusses an example for inter-process communication. In the 'GUI2VR' prototype outlined in section 6.2.5 socket-based communication between the GUI and the VR application is described.

6.2.1 GML 3 for describing spatiotemporal data[14]

GML 3 is applied to represent spatiotemporal LBM data generated by the TSS legacy system and to visualize dynamically the resultant larch defoliation. GML 3 (cf. subsection 3.3.2) specifies the GML 3 Schemas *temporal.xsd* and *dynamicFeatures.xsd* to represent temporal issues. The former schema defines primitives and properties for representing temporal instants and periods. The latter schema allows definition of elements and types to model dynamic features. A `DynamicFeature` (cf. Figure 6-4), aside from time-invariant properties, entails a `history` property to express the historical development of the feature. The `history` associates the feature with a sequence of time slices which include the dynamic properties of the feature (Lake et al., 2004).

[11] http://www.swisstopo.ch/en/basics/geo/system/refsystemCH (accessed April 18, 2006)
[12] Cited from Price et al. (submitted, p. 10).
[13] This section is based on section 5.1 of Isenegger et al. (2005).
[14] This subsection is based on section 5.2 of Isenegger et al. (2005).

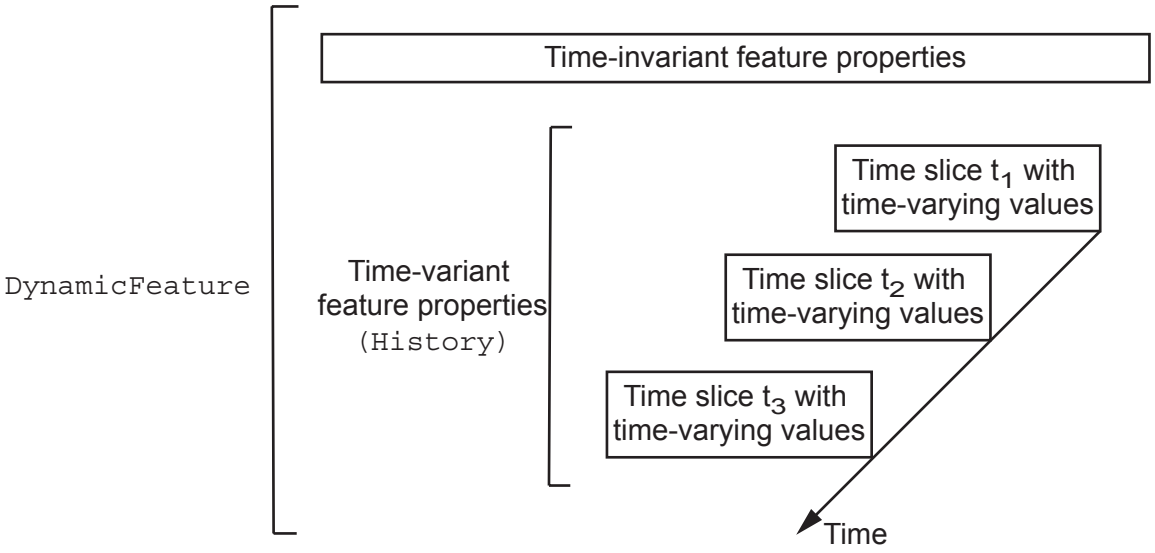

Figure 6-4 The `DynamicFeature` comprises besides of time-invariant properties a `history` property which may consist of a series of time slices with time-varying properties, after Lake et al. (2004).

Applying the spatiotemporal features of GML 3 to the LBM case study augments the expressivity of the data structure. The standard GIS representation of the LBM data is that for each year a dataset exists in a snapshot-like fashion containing the information about the study area (cf. Figure 6-5a). The temporal elements of GML 3 enrich the data structure to entail dynamic subsets of properties. A research area of the LBM case study is represented by a `DynamicFeature`. Time-invariant properties of a study area are for example its location, perimeter, and the coordinate system. The time slices comprise time-varying properties such as defoliation values, the amount of LBM larvae, and the year (cf. Figure 6-5c). The use of the temporal features of GML 3 leads to a more economic representation (Lake et al., 2004) due to the concentration of often voluminous (time-invariant) geographic data in only one place. On the other hand, this representation supports a more object-based view of the research area, which is mapped here as one object with time-invariant properties and series of time-varying properties.

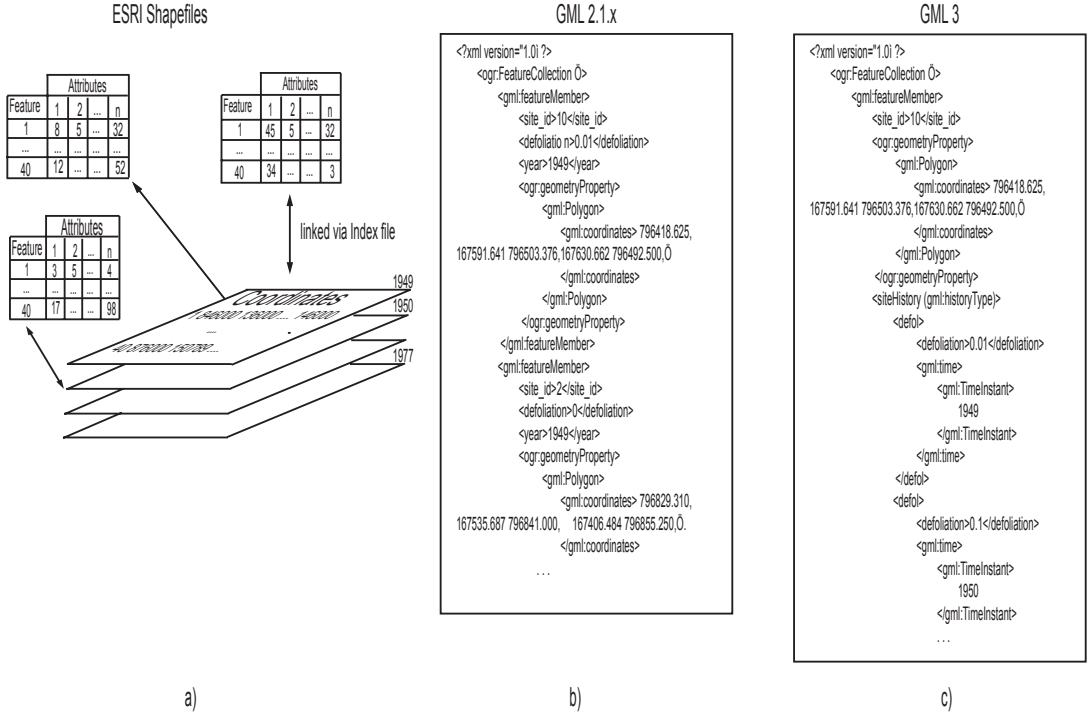

Figure 6-5 a) ESRI shapefiles showing areas with LBM infestation of the years 1949 to 1977. b) The same data encoded in GML 2.1.x. with repetitive encoding of time-invariant properties. c) In GML 3, opposed to GML 2.1.x, time-invariant properties are encoded only once while the time-variant properties are represented in `history` elements containing different values for each year (Isenegger et al., 2005).

The OGR[15] library of the GDAL module in the GRASS release 5.7 (Neteler and Mitasova, 2002) applied in the IPODLAS system provides the option to export data in the GML 2.1.x format. This option is used to generate a series of GML Documents and associated GML Schema. Each GML 2.1.x Document comprises besides other data the defoliation for each site for one year (cf. Figure 6-5b and Figure 6-6). To convert GML 2.1.x files into GML 3 files, the GML 2.1.x Documents and Schema files are parsed using the Python module *xml.dom*. Since in this conversion process the entire structure of the GML documents may have to be modified, the DOM-API was preferred to the SAX-API for parsing (cf. subsection 2.3.1). The main modification to the GML 2.1.x files to evolve them into the required dynamic GML 3 files is the addition of a dynamic element, i.e. the history element. The history element `<siteHistory (gml:historyType)>` (i.e. `siteHistory is` derived from the `gml:historyType`) is added to possibly carry the defoliation values of multiple years (cf. Figure 6-5c and Figure 6-6). The validity of the generated GML 3 Schema and Document was approved by applying XML Spy[16] (release 2005). Using the history element one GML 3 file can represent several years with associated defoliation values. This means that all GML 2.1.x Documents must be parsed and the respective year and defoliation values must be extracted and filled in the history

[15] OGR Simple Features Library (http://www.gdal.org/ogr/, accessed April 3, 2006) is a open source library providing access to a variety of vector file formats such as ESRI shapefiles and GML. OGR is the counterpart part of GDAL (http://www.gdal.org/, accessed April 3, 2006), which is a translator library for raster geospatial formats.

[16] XML Spy is a commercial XML editor, which provides validity checking for a XML Document against its associated XML Schema (http://www.altova.com/products_ide.html, accessed February 24, 2006).

element of the single GML 3 Document. The GML 3 structure exhibits a more object-centered nature: As Figure 6-5 shows, all information belonging to one object (which is here a site) can be bundled in one structure, i.e. the tree formed by the XML structure, and is not distributed in different "snapshots", each representing only one state in time. Moreover, the time-invariant data, e.g. the potentially complex geometry of the object, is not replicated over several files providing benefits for maintenance and disk usage.

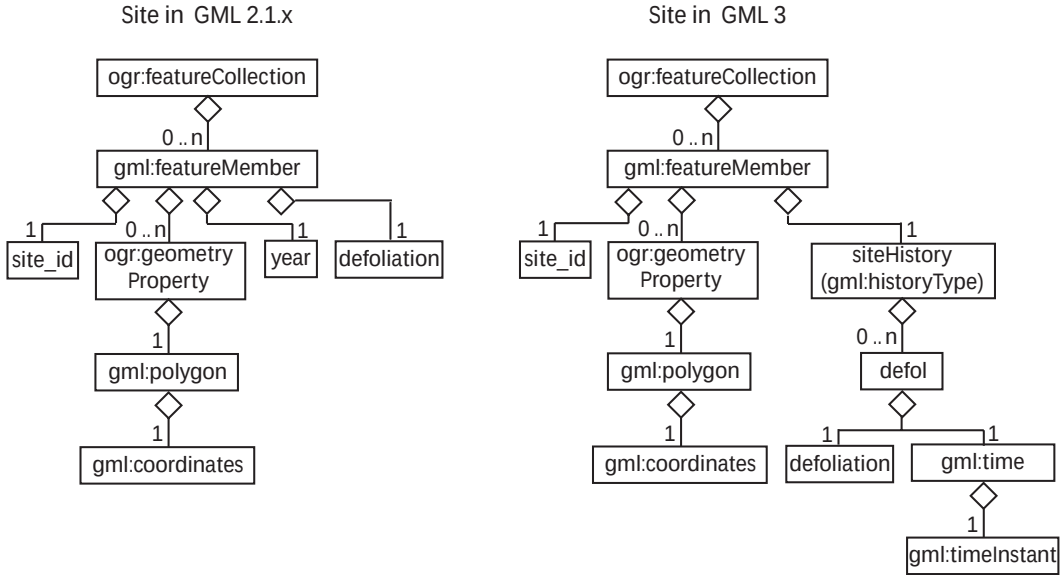

Figure 6-6 Comparison of site structure in GML 2.1.x (left) and GML 3 (right) in UML notation. In GML 2.1.x the site (site corresponds here to the element `featureMember`) consists of (besides `site_id` and `geometryProperty`) a `defoliation` value for one `year`, whereas in GML 3 the site can comprise in `siteHistory`, several elements `defol` each consisting of a `time` and `defoliation` value. The namespaces for which the elements are defined are given before the colon.

Since on the one hand there were no applications available which can import and visualize dynamic GML 3 data, and on the other hand SVG (cf. subsection 2.3.1) supports dynamic features, the (XML-based) GML 3 data is translated to SVG — the XML language for 2-D vector graphics — for visualization. In this use case the Java package javax.xml.transform is used to apply XSLT to convert the GML 3 Document 'sites.gml' into the SVG Document 'sites.svg' (cf. Figure 6-7). The XSL stylesheet 'gml2svg.xsl' defines templates which match in the GML Document the elements `gml:coordinates` and `defoliation`. Using this mechanism the coordinates of the polygons and the defoliation values of the respective years can be extracted from 'sites.gml'. In the SVG Document 'sites.svg' the extracted coordinates are scaled into the respective pixel values that fit to the screen and the sequence of defoliation values are matched with the the corresponding colors and can be displayed in an animation. The resultant dynamic SVG document can be viewed in most modern browsers using plug-ins[17].

[17] A well-known freely available SVG plug-in is the Adobe SVG Viewer (http://www.adobe.com/svg/viewer/install/main.html, accessed February 24, 2006).

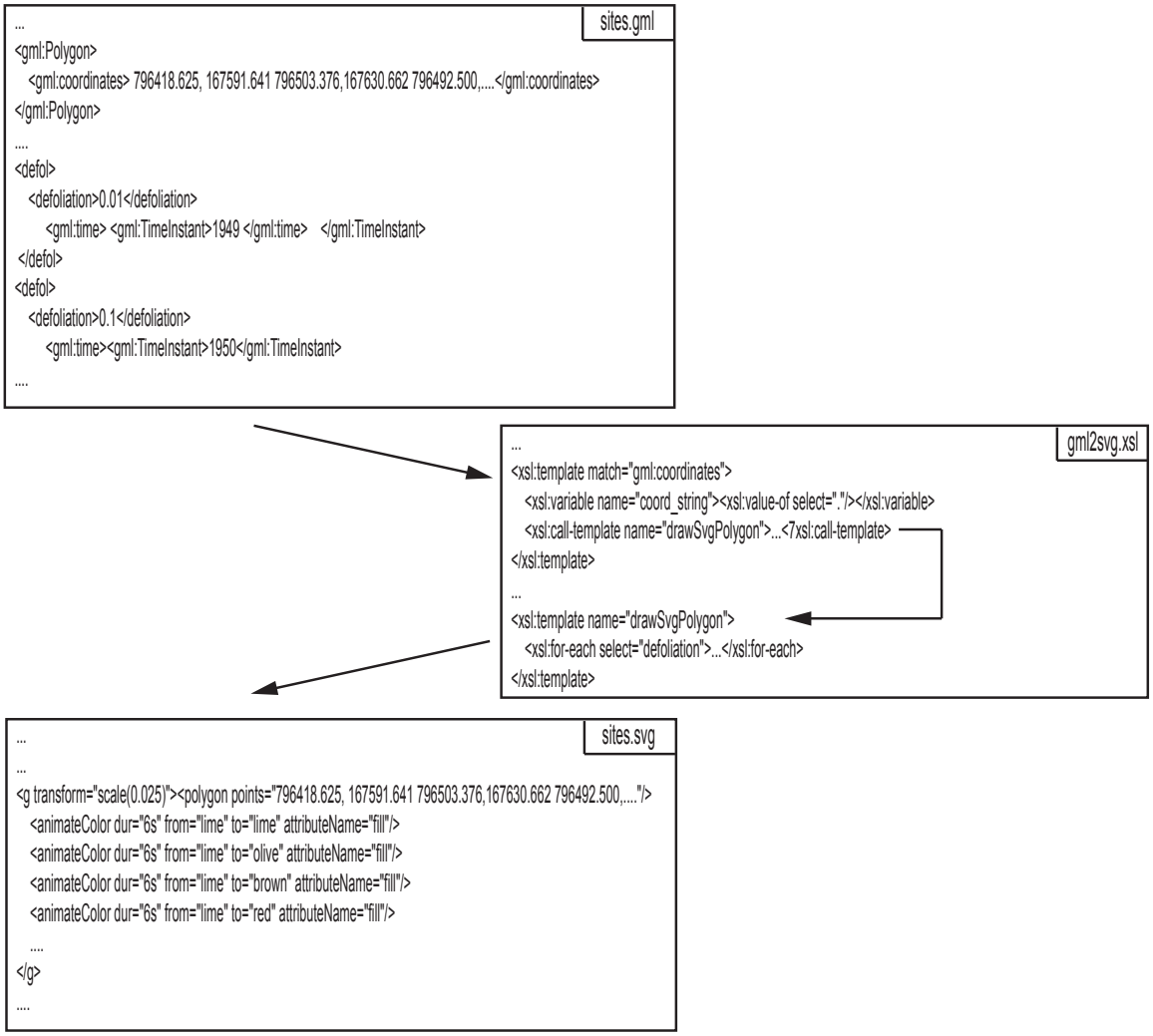

Figure 6-7 Application of the XSL stylesheet 'gml2svg.xsl' to generate the SVG file 'sites.svg' from the GML 3 file 'sites.gml'. The template `xsl:template match="gml:coordinates">` in 'gml2svg.xsl' finds the element `<gml:coordinates>` in 'sites.gml' and extracts the coordinates. Then, the template 'drawSvgPolygon' parses each defoliation value `defoliation` from 'sites.gml' and writes the (scaled) coordinates and the color associated with the defoliation value into 'sites.svg'.

6.2.2 The 'intelligent tree'

This preliminary version of the IPODLAS prototype was built to model the growth of an 'intelligent tree' (Fischlin et al., 2002a). The (virtual) tree was considered "intelligent" because it is aware of its location and therefore its growth conditions. In this prototype the user can specify the place where a tree may grow by selecting a location on the visualized surface with the pointing device within the VR subsystem (cf. Figure 6-8).

The GIS delivers data related to the habitat conditions (elevation, slope, and aspect) at the chosen location calculated from the DEM, while the TSS subsystem calculates the growth rate according to climate conditions, which are determined by elevation. Tree growth is visualized in a stepwise fashion in the VR subsystem. Data flow is handled by file exchange, while control flow is based on semaphores. Each subsystem is only allowed to access its respective input data file when the semaphore associated with this data file exists. Then the active subsystem has exclusive file access. After termination of all file accessing operations of the active subsystem the respective *Ready-semaphore* R_i associated with the output file of the active subsystem is generated to notify the other waiting subsystems. This preliminary prototype served as test bed for realizing a concrete division of labor among the subsystems and to test specific communication means such as file-coupling (Fischlin et al., 2002a). Exchanging information via files is an acceptable solution when advantages of simplicity outweigh performance losses. This depends on the operating system's characteristics of generating, reading, writing, and deleting files. A drawback is that synchronization of file access by semaphore files is not very flexible in comparison to a process-based approach with a central coordination process, i.e. synchronization of a more complex system by semaphores is quickly prohibitively challenging.

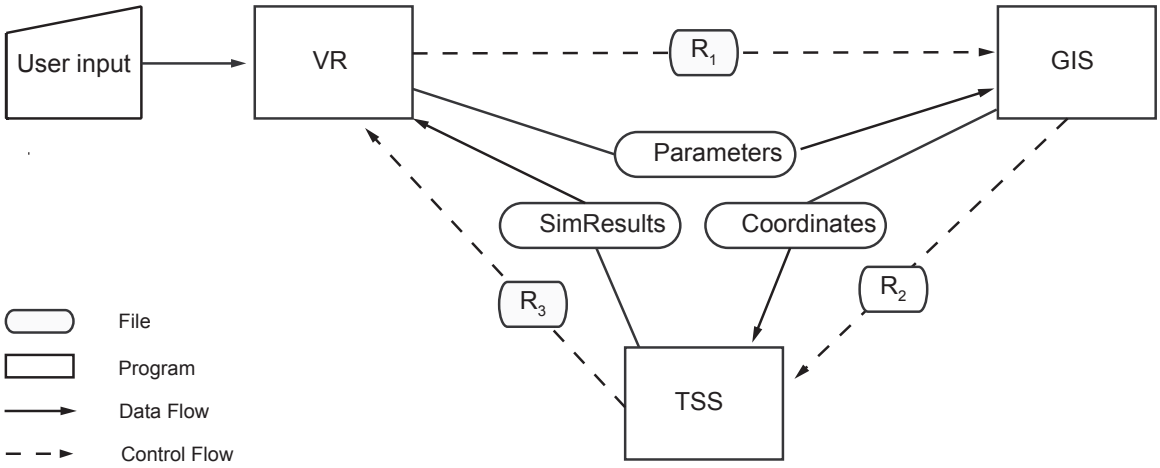

Figure 6-8 Overview of the "Intelligent Tree" Architecture. R_i (Ready-semaphore) represents the respective semaphore files. Parameters, Coordinates, and SimResults are the data exchange files (Isenegger et al., 2005).

6.2.3 Cross-implementation

In a subproject called as 'Cross-implementation' (Isenegger et al., 2004) some of the main concepts of an WLF simulation model already implemented in the GIS subsystem were implemented in the TSS subsystem, while crucial parts of an existing LBM simulation model already implemented in the TSS subsystem were implemented in the GIS subsystem. The WLF simulation model implemented in the TSS subsystem is *SPARKS* (Schöning, 1996), as LBM simulation model to be implemented in the GIS subsystem *LBM-M8* (Fischlin, 1982; Fischlin and Baltensweiler, 1979) was taken; a description of the simulation models can be found in Table 5-1. The aim of this subproject was to discover capabilities and limitations of the particular systems dealing with problems for which they are not originally designed. That is, the GIS was challenged with a simulation model computing mainly temporal processes and the TSS subsystem with processes with a strong spatial aspect. In this project the ArcInfo 8.1[18] Workstation software was used as

[18] http://www.esri.com/library/whitepapers/pdfs/arcgis_8.1.pdf (accessed April 5, 2006).

the GIS subsystem and RAMSES (Fischlin, 1991) as the TSS subsystem. While RAMSES provides libraries offering sophisticated mathematical and simulating capabilities, it lacks spatial functions, particularly for displaying geo-referenced data, spatial analysis, and storage of large volumes of spatial data. ArcInfo only has limited or no temporal functionality in comparison to RAMSES. Furthermore, besides its performance disadvantages the macro language *Arc Macro Language* (AML) (ESRI, 1993) does not support higher programming concepts. Aside from providing the somewhat trivial insight that applications deal best with the problems for which they are designed, this project highlighted needed functionality and which subsystem should best provide this functionality (Isenegger et al., 2004).

6.2.4 Remote Ramses

In the IPODLAS project Bergamin (2004) developed 'Remote Ramses' (cf. Figure 6-9), a client/server approach, where the client can remotly control the TSS legacy system, which acts as simulation server. RAMSES (Fischlin, 1991; Fischlin et al., 2002b) — described in subsection 3.1.4 — is the TSS legacy system applied in this preliminary IPODLAS prototype. 'Remote Ramses' acts as remotly controllable simulation server, which can load, compile, and execute simulation models and transmit the simulation results via network (Bergamin, 2004). Amongst other advantages RAMSES has been chosen as TSS subsystem due to RAMSES' interactive nature, e.g. in contrast to RASS (Thöny et al., 1995).

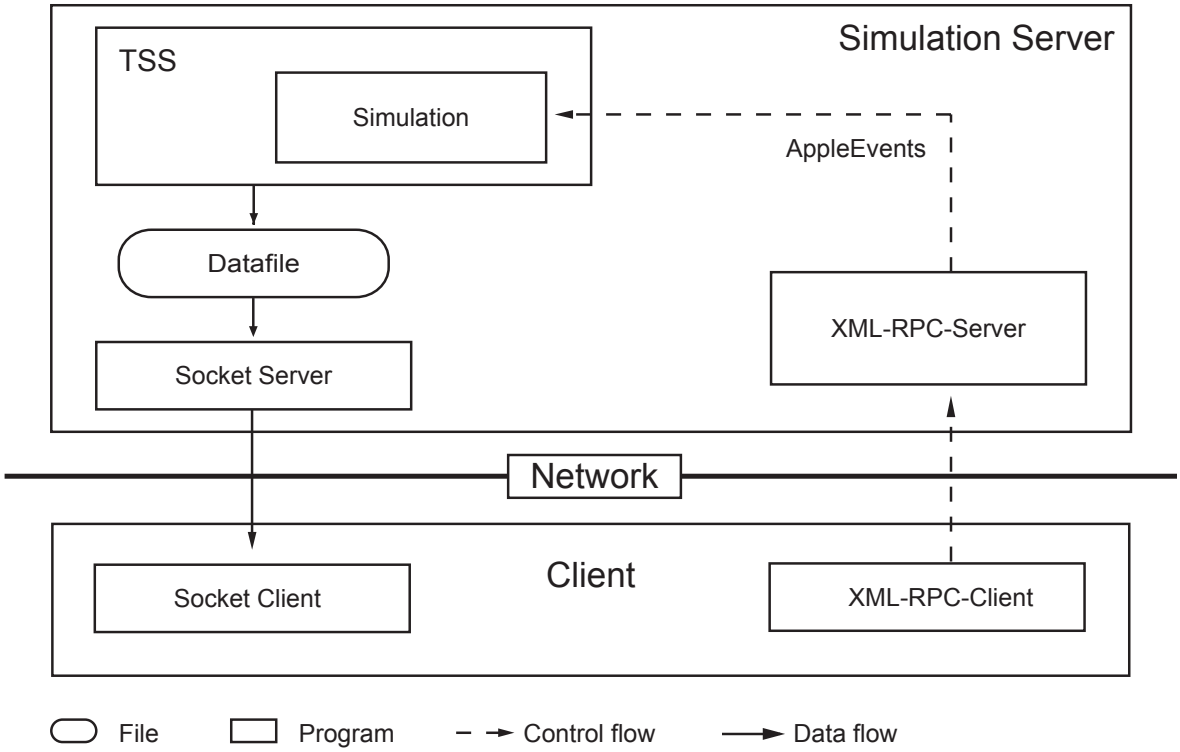

Figure 6-9 'Remote Ramses' (Bergamin, 2004), a socket client/server system to remotely control the TSS legacy system applied in the IPODLAS system, i.e. RAMSES (Fischlin, 1991), after Bergamin (2004).

The main drawback of RAMSES is its lacking supports of networked communication. Figure 6-9 describes the means applied in 'Remote Ramses' to control the TSS legacy system over a network applying the client/server approach. A client can trigger the TSS subsystem to start and stop simulation model runs interactively, change the active simulation model and transfer results encoded in XML (Bergamin, 2004). The inter-process communication between the XML-RPC-Server and the TSS subsystem uses AppleEvents, while the communication between the client and the server is done with sockets using the XML-RPC (UserLand Software, 2003) protocol.

6.2.5 The GUI2VR prototype

The 'GUI2VR' prototype implements a simple version of communication mechanisms applied in the final IPODLAS system described in chapter 7. Each subsystem implements both a socket client and a socket server establishing socket communication in a networked environment. In this example, the subsystems *GUI* and *VR* exchange information applying platform-independent communication mechanisms on the network and partly platform-dependent mechanisms within the platforms.

As described in section 5.2, the IPODLAS GUI has been applied as graphical definition of the state of the IPODLAS sytem in the respective action of a use case. The GUI has been developed on the Macintosh platform (10.3 and later releases) using Interface Builder[19] which is Apple's graphical user interface (GUI) builder for Mac OS X supporting graphical development by providing predefined widgets. A widget is component of a GUI the users interact with, such as windows, buttons, or menus. For instance, Figure 5-5 shows widgets of the IPODLAS GUI defined in the Interface Builder, which are used in the use case 'LE2' to start a LBM simulation applying the food hypothesis (Fischlin, 1982; Fischlin and Baltensweiler, 1979; Price, 2005) for the years 1949 to 1951.

The usage scenario of the 'GUI2VR' prototype is that of a user selecting on the IPODLAS GUI an animation of LBM migration of user-selected years in the Upper Engadine. Implmented on a Macintosh platform the GUI provides the functionality that user-triggered events in the GUI (e.g. a click on a button) can be catched as Apple events[20] in associated AppleScripts[21] (cf. Figure 6-10). Then the socket client on the GUI subsystem is invoked to request a socket connection to the associated socket server on the VR subsystem. When the connection is established, the socket client on the GUI subsystem sends the message containing the 'Start animation' information. After the socket server on the VR subsystem has received this information, it invokes the VR application to animate the LBM migration data of the user-selected years; in this prototype VTP (cf. subsection 3.24) is applied as VR application. When the animation has finished, the socket server on the VR subsystem notifies the socket client on the GUI subsystem about the completion of the animation. The socket client invokes a Python

[19] http://developer.apple.com/tools/interfacebuilder.html (accessed February 27, 2006).

[20] "An Apple event is a type of interprocess message that can specify complex operations and data. Apple events allow you to gather all the data necessary to accomplish a high level task into a single package that can be passed across process boundaries, evaluated, and returned with results. The Mac OS uses Apple events to communicate with applications. Apple events are also an essential part of the AppleScript scripting system […], applications […] can respond to a variety of Apple events by performing operations or supplying data." (http://developer.apple.com/documentation/AppleScript/Conceptual/AppleEvents/intro_aepg/chapter_1_section_1.html, accessed February 27, 2006).

[21] AppleScript is a scripting language built into the Mac Os X offering the user mechanisms to control the system and exchange information between applications (http://www.apple.com/macosx/features/applescript/, accessed February 27, 2006).

script to create an AppleEvent telling the GUI to generate an Alert Box, which informs the user about the completion of the animation.

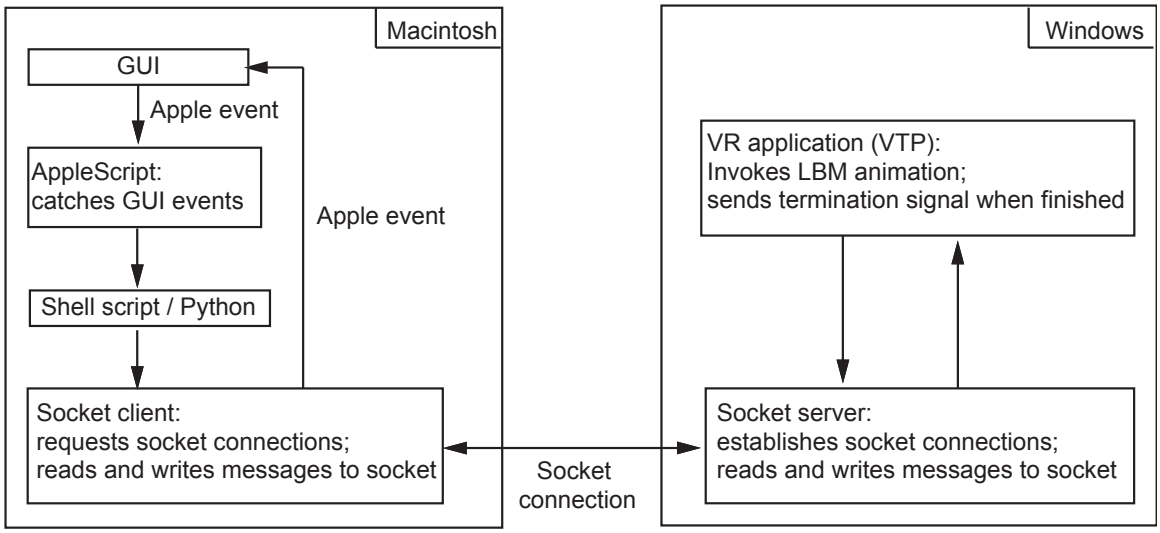

Figure 6-10 The 'GUI2VR' prototype illustrates the communication between the IPODLAS GUI on the GUI subsystem (on a Macintosh platform) and the VR application on the VR subsystem (on a Windows platform). For communication within the platforms partly platform-dependent communication mechanisms are used (e.g. Apple events), the communication between platforms is established applying sockets.

7 The IPODLAS system

This chapter gathers *design considerations* about the functionality of the IPODLAS system discussed in chapter 5 and insights gained from the preliminary prototypes described in chapter 6 and derives from there the resulting *final IPODLAS system*. In section 7.1 concepts and considerations used to develop the user interface are described on an abstract level: the goal is not to define explicitly the widgets of a graphical user interface, but to show which type of user interface the IPODLAS system requires. Section 7.2 applies the specifications collected in the use cases described in section 5.2 and derives constraints and requirements which lead to the software architecture of the IPODLAS system. The interaction and communication model of the IPODLAS system are explained in section 7.3 on the example of the implementation of the use case 'LE3 ext'. The run time characteristics of the IPODLAS sytem are discussed in section 7.4 on the implementation of the use case 'LE3 ext'.

7.1 User interface design

Many computer systems are built to interact with people. The user interface (UI) of the computer provides the functionality for the user to interact with the system. One aim of designers of the computer's UI is to support the users to fulfill their tasks in the most efficient and convenient way. The more the designers know about the prospective users and their tasks, the better the UI can be designed to satisfy the users' requirements (Rubin, 1988).

Figure 7-1 shows the levels of user interface design identified by Löwgren (1993). In the case of the IPODLAS system, the UI design refers to the design of the *IPODLAS GUI*. In subsection 7.1.1 the design level 'System services' of the IPODLAS GUI is described focusing on how different users see the IPODLAS system due to their different tasks. In subsection 7.1.2 the subsequent design level 'User's model and metaphor' of the IPODLAS GUI is explained. The IPODLAS GUI was mainly developed to support the definition of the use cases described in section 5.2. Hence, the corresponding GUI functionality has been implemented only for a part of the graphical widgets designed in the IPODLAS GUI. Consequently, the more concrete and specific levels of the user interface design levels specified by Löwgren (1993) are likely to be subject to change and are thus not discussed here.

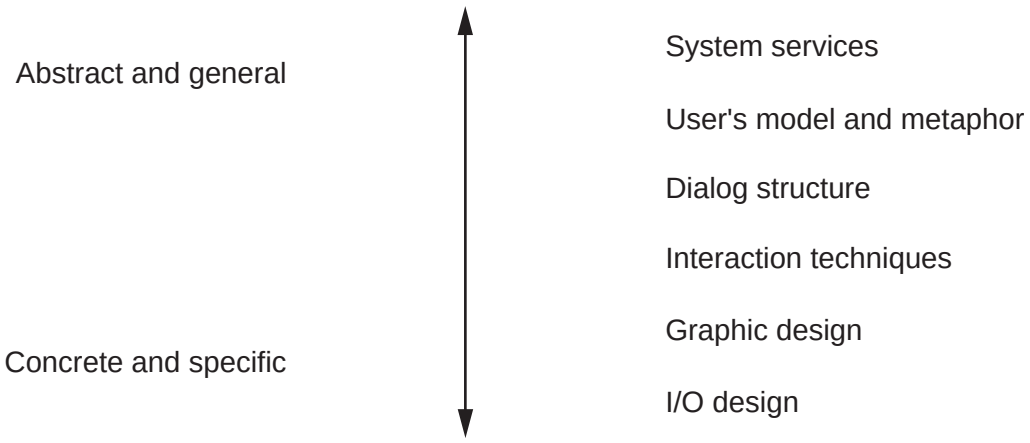

Figure 7-1 User interface design levels, after Löwgren (1993).

7.1.1 System services

The different types of users specified in section 5.2 are used as archetypes of potential IPODLAS user to design the IPODLAS system; their behavior and requirements are exemplars for the behavior and requirements of entire classes of potential IPODLAS users.

The different types of users apply the IPODLAS system for a variety of tasks. The users' tasks constrain the system services of the IPODLAS system the users want to apply. Thus, as Figure 7-2 illustrates, the tasks of the users form the users' view of the IPODLAS system.

The *pilot user* perceives the IPODLAS system as an information discovery system, which may support him to find the information she/he is looking for. The activities of this user aim at retrieving information and not at generating new information. Thus, the pilot user's normal behavior is restricted to explore the virtual scenery and to browse for information about the phenomena of interest. Besides just navigating and selecting data, she/he can obtain desired information by running stored animations or executing simulation models applying mostly default settings. Since the pilot user is not an expert knowing these simulation models very well, she/he normally just keeps the default configurations.

The *expert user* may conceive the IPODLAS system also as a research instrument. Besides information retrieval (the main type of functionality the pilot user applies), the expert user can generate new information using the functionality of the IPODLAS system. She/he is expected to change model parameters and to compare generated simulation results, since she/he is able to asses the consequences of changing model parameters. Moreover, the expert user can generate new simulation models by coupling models or by introducing new models.

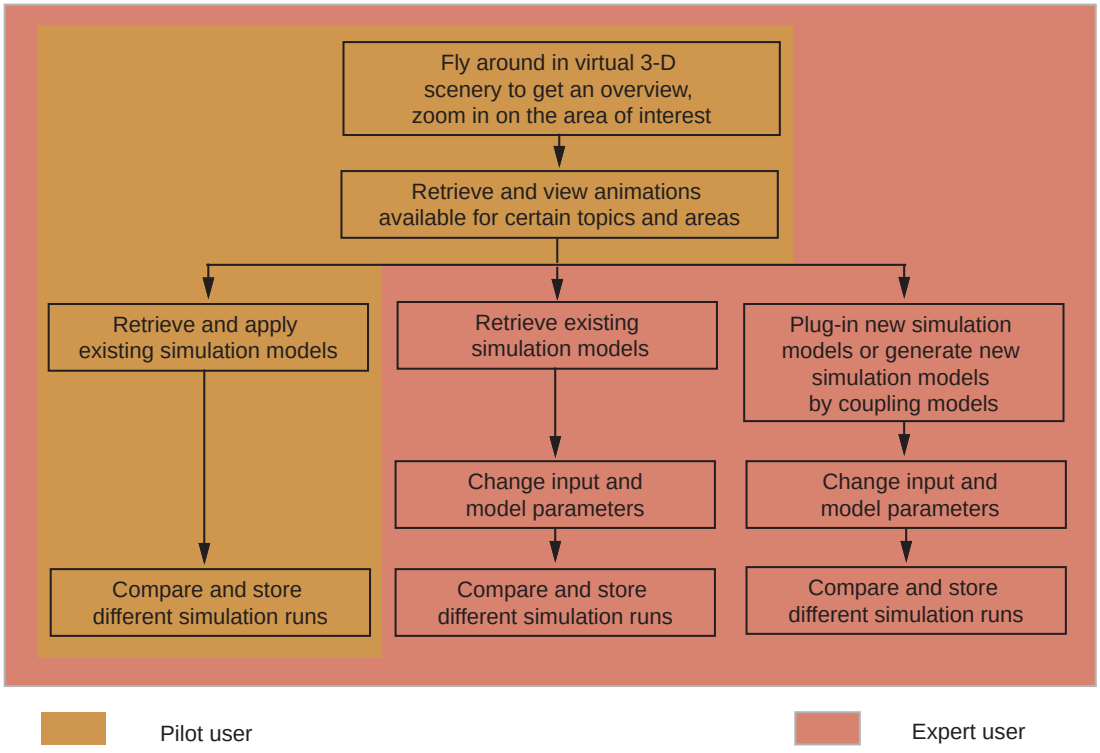

Figure 7-2 The different type of users apply different types and ranges of functionality of the IPODLAS system. Thus, by applying different ranges of functionality the users perceive the IPODLAS system in different manners, after Bergamin (2003).

7.1.2 User's model and metaphor

Rubin (1988) defined the *user model* as the model the designer has of the prospective user, while the *user's model* represents the model that the user has in his mind of the UI, obtained through previous experience, interacting, or literature. Consequently, there are several user's models depending on the task of the user and her/his knowledge about the UI.

The state or the mode[1] of the user interface is defined in Barfield (1993) through the set of operations a user can execute; the operation(s) needed to get from one state to another state are denoted as transitions. In interactive systems, the UI reacts to user interaction, i.e. states are changed by user interaction. A UI is modal, if it exhibits more than one mode and it is denoted as modeless if it only adopts one (Barfield, 1993).

The user's model is an abstraction of the real computer system the user has in mind, i.e. a simplification containing only what is necessary for the user to fulfill her/his tasks. It tells the user how to get from one state to the next state using a certain transition (Barfield, 1993; Rubin, 1988). To assist the user in obtaining an appropriate user's model, Barfield (1993) suggests three principles: Firstly, the structure of the UI can be explained by exploiting *previous experience* of the user. The UI is designed so that it imitates features from known user's models or by using a *metaphor*, where the user can deduct from properties of the metaphor to properties of the UI. An example for this is the 'desktop metaphor' (Brooks, 1987) of today's workstations. The application of metaphors inherently bears the danger of the 'too much / too little problem' (Löwgren, 1993): the metaphor can infer too much, so that actions possible within the metaphor cannot be executed in the real UI or the metaphor can infer too little, i.e. the metaphor shows the user not all possibilities of the UI. Secondly, *interaction* of the user with the UI must always be prompted with feedback providing the user with information about the UI's state and thirdly, the interaction of the user with the UI must be *observable*, the user must see which interaction causes which UI reaction.

Design of IPODLAS GUI

In the design of the IPODLAS GUI a general principle applied was that the GUI should be as simple as possible, but expressive enough to support the users to fulfill their tasks; this refers to the usability[2] of the GUI. Applied to modality of the UI, this means, that too many modes tend to confuse and overhelm the users, whereas a modeless UI provides no user guidance. The metaphor used in the development of the IPODLAS GUI for the pilot user was the one of a flight simulator. The user sits in the cockpit and sees on the main screen the landscape she/he is flying over. She/he is in control of the spatial navigation and sees the landscape in 2-D or in 3-D depending on the flying altitude visualized. In addition, the expert user can access and control animations or simulations via extra widgets, such as extra menus or windows provided by the IPODLAS GUI.

State diagrams help to define the user's models: it defines the states the UI can adopt and which operations are available for the respective state. The state diagram illustrated in Figure 7-3 represents the major states of the IPODLAS GUI and the associated operations on a rather general level; Table 7-1 specifies the states and

[1] A mode "is a subset of the set of commands, consisting of those commands that are active at a given moment" (Nievergelt and Weydert, 1980, p. 327).

[2] The document 'ISO 9241-11 (1998) Guidance on Usability' (http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16883&ICS1=13&ICS2=180&ICS3=, accessed May 5, 2006) defines usability as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use".

associated operations, which are depicted in the state diagram in Figure 7-3. The state diagram illustrates that after starting the user opens an existing project or generates a new one. All information associated with the current usage of the IPODLAS system is to be maintained in the current project. After selecting a topic the user can access and control the provided animations and simulations. In each of the states 'Project', 'TopicSelected', 'AniRunning,'SimuRunning', 'AniPaused', and 'SimuPaused' spatial navigation and several configuration operations are supported by the GUI. It is possible to save or exit the project in each state; in the state 'AniRunning' / 'SimuRunning' the animation or simulation must be stopped first.

By defining the possible operations of a user at each state of the system the state diagram provides constraints for the development of the whole IPODLAS system. The state diagram defines on the one hand the information presented to the user by the system in each state of the system and on the other hand the interaction possibilities, i.e. the possible operations of the user at this state. For the designer of the GUI the state diagram helps to define which information are required by the user to fulfill their task. For the system developper the state diagram defines which operations a user can trigger at which state of the GUI and thus which operations must expected and treated by the rest of the IPODLAS system.

The state diagram shows that the states can be classified into different groups of states according to the types of operations that can be iniated by the user at the respective state of the UI:

- the state 'OFF': only the operation 'startUp' must be handled by the IPODLAS system
- the states 'NoProject', 'SaveProject1', 'SaveProject2', 'SaveProject3': only file dialog operations must be handled by the IPODLAS system
- the states 'Project', 'TopicSelected', 'AniRunning'/'SimuRunning', 'AniPaused'/'SimuPaused': besides file dialog, spatial navigation and view configuration operations must be handled by the IPODLAS system
    - the subgroup 'TopicSelected', 'AniRunning'/'SimuRunning': animation/simulation operation must be handled by the IPODLAS system

-

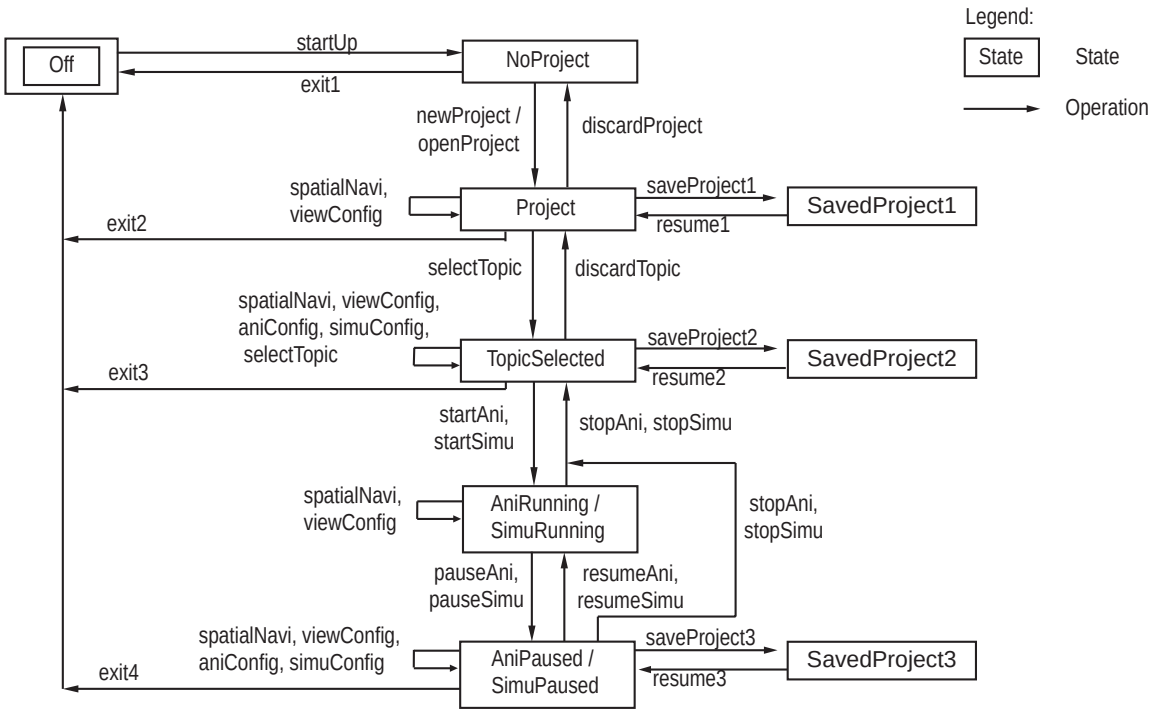

Figure 7-3 The state diagram showing states and associated operations of the IPODLAS GUI. States and operations are detailed in Table 7-1.

State	Description of state	Available operation	Description of operation
Off	The IPODLAS system is shut down	startUp	
NoProject	The IPODLAS system is up, but no project selected	newProject	New project generated
		openProject	Opens a existing project
		exit1	Shuts down the IPODLAS system
Project	A project has been generated or selected: the project is identifiable by its name now. In this state 2-D and 3-D spatial navigation is possible	selectTopic	Selects topic of interest
		discardProject	Discards project
		saveProject1	Saves project
		spatialNavi	Spatial navigation: in 2-D, 3-D, or both
		viewConfig	Configures view: e.g. select additional data to display
		exit2	Shuts down the IPODLAS system
SavedProject1	A project has been saved	resume1	Resumes project
TopicSelected	A topic within the project is selected	discardTopic	Information concerning this topic is discarded
		saveProject2	Saves project in current state
		spatialNavi	Spatial navigation: in 2-D, 3-D, or both
		viewConfig	Configures view: e.g. select additional data to display
		aniConfig	Selects and configures animation(s): e.g. start and stop time, time step

		simuConfig	Selects and configures simulation(s): e.g. start and stop time, monitoring variable(s), time step
		selectTopic	Selects topic of interest
		exit3	Shuts down the IPODLAS system
		startAni	User starts animation
		startSimu	User starts simulation
SavedProject2	A project has been saved in current state	resume2	Resumes project in current state
AniRunning / SimuRunning	An animation/simulation is running	stopAni	Stops animation
		stopSimu	Stops simulation
		pauseAni	Pauses animation
		pauseSimu	Pauses simulation
		spatialNavi	Spatial navigation: in 2-D, 3-D, or both
		viewConfig	Configures view: e.g. select additional data to display
AniPaused / SimuPaused	An animation/simulation is paused	stopAni	Stops animation
		stopSimu	Stops simulation
		resumeSimu	Resumes animation/simulation
		saveProject3	Saves project in current state
		spatialNavi	Spatial navigation: in 2-D, 3-D, or both
		viewConfig	Configures view: e.g. select additional data to display
		aniConfig	Configures animation: e.g. start and stop time, time step
		simuConfig	Configures simulation: e.g. start and stop time, monitoring variable(s), time step
		exit4	Shuts down the IPODLAS system
SavedProject3	A project with selected topic and a paused animation/simulation has been saved	resume3	Resumes project in current state

Table 7-1 States and available operations for the respective states of the IPODLAS GUI specified in Figure 7-3.

7.2 Software architecture

The overall goal is to develop an architecture that makes the system resilient to change or change tolerant. The software architecture includes the most important static and dynamic aspects of the design of a system. It focuses on significant structural elements as well as on the interactions that occur among these elements via interfaces (Jacobson et al., 1999)[3].

[3] This section is based on section 5.2 of Isenegger et al. (2005).

7.2.1 Nonfunctional requirements

The nonfunctional requirements (cf. section 2.1) result from demands about the characteristics of the IPODLAS system that are not explicitely specified in the use cases, but are nevertheless perceived as benefical properties. Some nonfunctional requirements can be derived from analysis of usage scenarios collected in the use case model, some come from general considerations of the designers about the software architecture.

The use cases described in subsection 5.2 specify the IPODLAS system as a user-driven, interactive, and real-timesystem (cf. Table 7-2). In a *user-driven* system each action of the user causes an action in the IPODLAS GUI which may trigger subsequent actions in the IPODLAS system. *Interactivity* is an important requirement to make the IPODLAS system user-driven. An user can navigate in a virtual scenery provided by the VR legacy system and therefore interact with the IPODLAS system. She/he can control the IPODLAS system via the GUI, e.g. selecting topics of interest, starting and stopping animations and simulations, etc. In the subsystems of the IPODLAS system interactivity is — besides in VR — also a beneficial property of RAMSES (cf. subsection 3.4.1) (Fischlin, 1991) which is the representative legacy system of TSS in the IPODLAS system. For instance, RAMSES must be interactive to be able to pause and resume simulations and to change settings of parameters and values during a simulation run. In GIS, the main usage mode involves the interactive application of GIS functionality. *Real-time behavior* is required to make the navigation in the IPODLAS system user-friendly. This means in particular that the IPODLAS system must let react the VR subsystem, i.e. the VR legacy system, immediately upon user input in order to establish the known movie-like 'look and feel' of VR (Van Dam et al., 2000). Interactivity and real-time behavior requires asynchronous communication (cf. subsection 2.4.1). When the user starts a simulation, which may take an hour, the user does not want to wait for the end of the simulation to do her/his next action. Instead, the IPODLAS system starts the simulation, notifies the user about this event and may estimate the duration of the simulation, and then wait for the next user interaction. When the simulation is finished the user is notified about this event.

Nonfunctional requirement	Description
User-driven	The system's activity is controlled by the user.
Interactivity	"[…] each user entry causes a response from or action by the system." (IEEE, 1990, p. 41)
Real-time behavior	The system can " […] control, monitor, or respond in a timely manner to a external process." (IEEE, 1990, p. 61)

Table 7-2 Nonfunctional requirements derived from the analysis of use cases.

Other nonfunctional requirements are aspired by the designers of the IPODLAS system to facilitate the development of the IPODLAS system and its maintenance (cf. Table 7-3); these are maintainability, extensibility, scalability, and conceptual independency of the IPODLAS system from any particular subsystem or platform (Allgöwer et al., 2001). An *maintainable* and *extensible* system architecture considers and facilitates future modifactions by containing the consequences of changes to a limited part of the system and not having to change major structures of the architecture. Extensions can be made by modification of existing functionality or by addition of new functionality (Jacobson et al., 1999). *Scalability* can be seen in a similar context; a system scales well when the growth of the system, e.g. by addition of other components, does not have major impacts on system properties, e.g. the performance (Ghezzi et al., 2003). *Conceptual independence* of the IPODLAS system from any particular subsystem or platform means that the IPODLAS

system should not be conceptually dependent on the use of a certain TSS, VR, or GIS legacy system, but that the components are interchangeable. The goal of the IPODLAS framework is to develop concepts and interfaces not limited to particular legacy systems. Thus, the IPODLAS system requires a design not restricted to specific platforms, languages, or operating systems (Allgöwer et al., 2001). In the same category falls the application of *open source software* to develop the IPODLAS system. Using non-proprietary software enables the IPODLAS project team to obtain and modify the source code according to its needs without paying royalities or fees.

Nonfunctional requirement	Description
Maintainability	"The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment." (IEEE, 1990, p. 46)
Extensibility	"The ease with which a system […] can be modified to increase its storage or functional capacity." (IEEE, 1990, p. 32)
Scalability	The ease with which a system can be modified by adding new resources (I.Sommerville, 2001).
Conceptual independence	Independence of the design of any particular legacy system
Use of open source software	Application of open source software

Table 7-3 Nonfunctional requirements required to support the development and maintenance of the IPODLAS system.

7.2.2 Combination strategy

As stated in chapter 1, one aim of the IPODLAS framework is to offer landscape visualization, analysis, and modeling functionality, and not to reinvent TSS, VR, or GIS functionality. To use the functionalities of TSS, VR, and GIS applications together, the IPODLAS system must be able to access their functionalities and provide them to the user. The TSS, VR, and GIS applications are integrated as legacy systems into the IPODLAS system and form the TSS, VR, and GIS subsystem. Figure 7-4 shows the architecture of the IPODLAS system: all subsystems are connected via the kernel and can be accessed via network providing their functionality to the user. The subsystems TSS, VR, and GIS can be accessed and controlled by a common UI and share a common storage.

Central synchronization and storage

As the use cases and the listings of functionality (cf. chapter 5) demonstrated, the interactions of the subsystems follow a certain sequence of (inter)actions that require synchronization. The IPODLAS system applies the *blackboard architecture* (cf. 2.2.1) by designating a subsystem to act as "blackboard", where the other subsystem exchange information. In the IPODLAS system the *IPODLAS kernel* or just *kernel* is the only interface establishing the communication between the subsystems. Thus, the kernel can provide the synchronization between the subsystems by managing the control flow. The IPODLAS system provides a mapping of the necessary parts of the data models of the respective legacy systems.

The *mediating* kernel waits for requests coming from the GUI, analyzes them, and, if necessary, breaks them down into subrequests and dispatches the subrequests to the appropriate subsystems. In parallel, the kernel provides feedback notifying the requesting subsystem that the request has been received. The result of the request is propagated to

the appropriate subsystem. All the control flow and the data flow is passing through the kernel.

A common storage is beneficial to prevent data heterogeneities and to limit the amount of data exchange (cf. subsection 2.4.1). The *IPODLAS storage* holds data that may be useful for all subsystems and metadata describing the data available to the IPODLAS system. The IPODLAS storage is only accessible through the kernel. The GIS storage is the second persistent storage of the IPODLAS system; it is managed by the GIS legacy system in the GIS subsystem. In the current prototype not all data required by subsystems of the IPODLAS system is hold in the central IPODLAS storage; typically data which is only accessed by one subsystem is stored local in the respective subsystem, e.g. auxiliary data needed by the TSS subsystem for temporal simulation or image data required by the VR subsystem for photorealistic visualization.

Communication

The communication on the technical level is established by sockets between subsystems and by various, possibly platform-specific mechanisms within subsystems. The communication *between subsystems* of the IPODLAS system is realized by socket clients deployed on each subsystem establishing a (platform-independent) socket connection to the socket server deployed on the kernel subsystem. The basic socket communication functionality is provided by *TwistedMatrix* (Fettig, 2005) (cf. subsection 2.4.3), a framework for programming network services and applications. TwistedMatrix supports the concept of *nonblocking asynchronous servers*. This means that on each subsystem read and write operations on the socket interface can be executed asynchronously without blocking other subsystem tasks. This is realized by allocating for each socket client an individual instance of the socket server, which manages the communication with its associated socket client. *Within subsystems* of the IPODLAS system, which reside on one platform, the respective appropriate mechanisms are applied, e.g. sockets, file exchange, or program invocation.

On the conceptual level, the required asynchronous nature of the communication between the subsystems is established by message passing (cf. subsection 2.4.1). A subsystem sends a request as a message to another subsystem using the socket connections and may continue without waiting for the result of the request. The messages exchanged define the communication protocol, i.e. the respective set of commands that the subsystems must support.

Modularity

Modularity of the software architecture is enhanced by the synchronization subsystem — the IPODLAS kernel — limiting the number of interfaces needed for communication between the subsystems. The role of the kernel as the only communication interface between the subsystems means that changes in the interaction of the subsystems or even the exchange of a subsystem, for example the use of another GIS, need only be implemented in the IPODLAS kernel.

Due to the iterative nature of software development, in some chapters of this thesis the individual steps of architecture refinement are split into the different phases of development (e.g. early phase, advanced phase). Owing to the modular design of the IPODLAS system, the dependencies within the IPODLAS system are limited and therefore when supporting the same interfaces the subsystems in different phases can interact smoothly and seamlessly with another. This means that advances of certain aspects of the software architecture can be implemented while side effects are limited by the modular nature of the IPODLAS system.

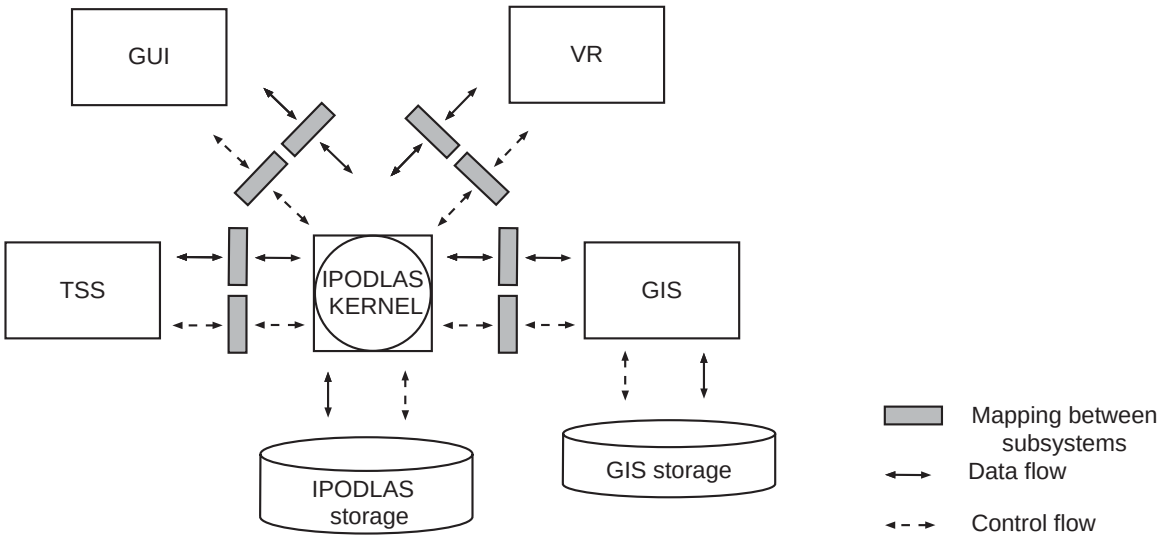

Figure 7-4 A schematic view of the IPODLAS system software architecture, after Isenegger et al. (2005), combining the subsystems TSS, VR, GIS, GUI, IPODLAS storage, GIS storage, and the IPODLAS kernel. The kernel is the central synchronization subsystem handling the control flow and the data flow between the subsystems through mapping of the required parts of data models and of the functionalities.

7.2.3 Layout of the IPODLAS system

While in subsection 7.2.2 constraints and requirements and the resulting software architecture of the IPODLAS system are discussed on a general level, this subsection tries to go more into the details of the software architecture and of the communication between the subsystems. In this subsection the composition and deployment of the IPODLAS system is described. Furthermore, crucial issues of elements specifying the communication within the IPODLAS system are addressed.

Composition and deployment

Figure 7-5 illustrates the composition and configuration of the main classes of the IPODLAS system. Classes defined within the TwistedMatrix framework (Fettig, 2005) (cf. subsection 2.4.3) provide basic network communication funtionality; to exploit their fucntionality from these classes the principal IPODLAS system classes have been derived. All classes of the IPODLAS system are derived from `Internet.Reactor` which establishes communication through socket connections and provides the polling mechanism checking for the state of socket connections for each instance of `Internet.Reactor`. The class `Protocol` implements several interent protocols, such as http, telnet, and ftp. A problem of transmitting data over socket connections is use of delimiters. When data is transmitted through a socket connection, it cannot be guaranted that the whole dataset or not more than one dataset has been transmitted or received, respectively. Using delimiters the TwistedMatrix class `LineOnlyReceiver` ensures that only complete datasets, i.e. all data enclosed between two delimiters is exchanged between the subsystems. `IpodlasKernelfactory` and `IpodlasClientFactory` are children of `Factory` used to instantiate *`IpodlasKernel`* (`IK`) and *`IpodlasClient`* (`IC`). `IpodlasClientFactory` is derived from `ReconnectingClientFactory` (in Figure 7-5 abbreviated as `ReconClientFactory`) which provides the functionality of reconnecting broken socket connections.

To communicate selective with a particular subsystem, the IPODLAS system, in particular the IPODLAS kernel, have to know which `IK` instance communicates with

which IC instance deployed on the subsystem. By linking each IK instance with its associated IC instance, the hash table[4] clientsAtSubsys in IpodlasKernelfactory records which instance of IK communicates with which IC instance deployed on the subsystems.

The class IpodlasMessage provides the class variable ipodlasMessageTable used to represent the messages ipodlasMessage (IM), which are exchanged between the IK and IC instances in the IPODLAS system. The class IpodlasMessage offers functions for handling the IMs, for example the parsing of IMs.

The ipodlasMessageTable is designed as class attribute, so that all instances of the class IpodlasMessage, i.e. in particular its subclasses IK and IC, can access ipodlasMessageTable. ipodlasMessageTable is implemented as hash table storing all generated IMs. The IK instances have access to this hash table and can retrieve information about a certain IM using its key. The IK and IC class handle the IMs received with functions lineReceived() in their central control functions kernelControl() and clientControl(), respectively.

[4] In the data structure hash table or hash map keys are associated with values: given a key, the associated values can be found (Knuth, 1997). In Python, the data type dictionary is used to represent hash tables (Martelli, 2003).

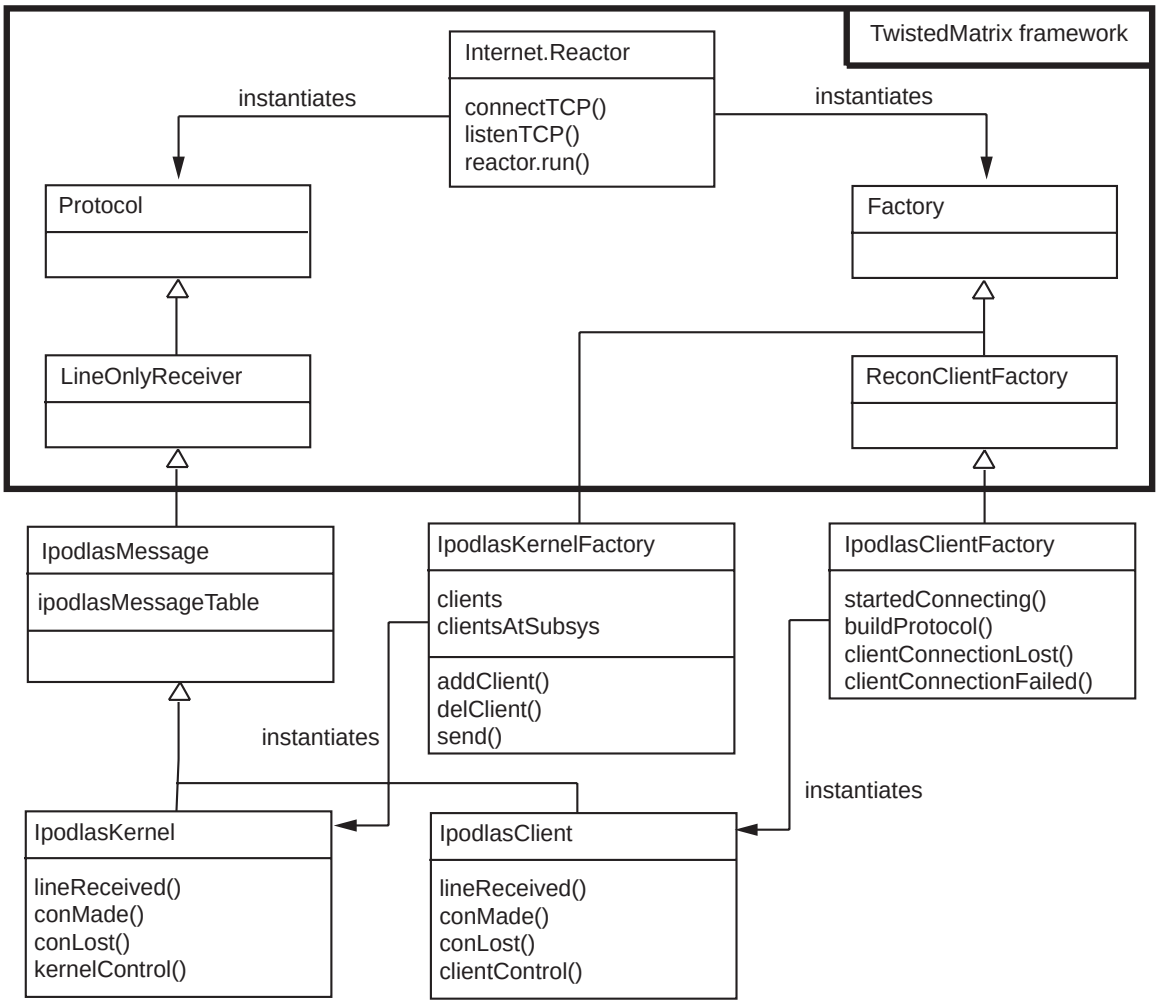

Figure 7-5 Simplified UML class diagram of the IPODLAS system. The classes within the bold rectangle are classes defined in the TwistedMatrix framework (Fettig, 2005). The classes below are derived classes which establish the IPODLAS system.

The deployment of the elements of the IPODLAS system is shown in Figure 7-6. On each subsystem (except the kernel subsystem) an `IC` instance establishes a socket connection to its associated `IK` instance on the kernel subsystem. All `IC` and `IK` instances use the polling mechanism of TwistedMatrix to check if a read or write request occurs on their respective socket connection. Within the subsystems various communication mechanisms are used in the IPODLAS system. The GUI and the VR subsystem use the communication mechanisms described in the 'GUI2VR' prototype (cf. subsection 6.2.4). This means using AppleEvents, AppleScripts, shell scripts, and Python on the GUI subsystem, which is deployed in the final IPODLAS system on a Macintosh platform, and socket connections and C programs on the VR subsystem, which is deployed on a Windows platform, respectively. The communication with the TSS legacy system — in the final IPODLAS system this is RAMSES (Fischlin, 1982) — is established using the communication solution implemented in 'Remote Ramses' (Bergamin, 2004) (cf. subsection 6.2.3) using AppleEvents and sockets. As GIS representative in the final IPODLAS system GRASS was chosen (Neteler and Mitasova, 2002) (cf. subsection 3.3.4); Python and shell scripts are applied for the communication between the `IC` and GRASS on the GIS subsystem.

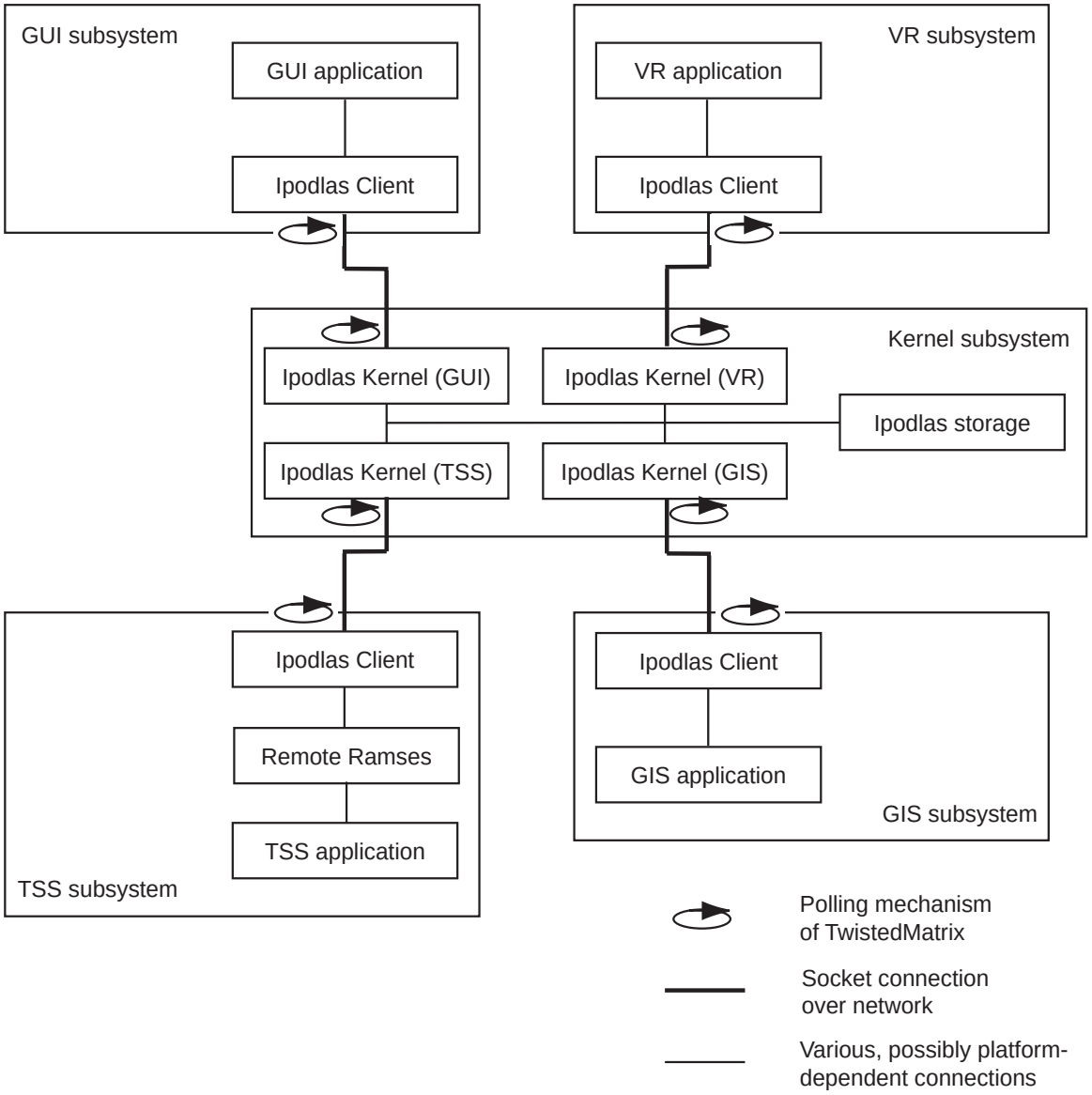

Figure 7-6 Deployment of the IPODLAS system. On the subsystems TSS, VR, GIS, and GUI an `IpodlasClient (IC)` instance establishes the communication with its associated `IpodlasKernel (IK)` instance on the Kernel subsystem over sockets.

The ipodlasMessage (IM)

The class `IpodlasMessage` (cf. Figure 7-5) provides a class attribute `ipodlasMessageTable`. The `ipodlasMessageTable` is implemented as a hash table exhibiting a key and the data structure `ipodlasMessage (IM)` in XML code as a value. `ipodlasMessageTable` is a class attribute, meaning that there is only one instance of `ipodlasMessageTable` and so all instances of the class `IpodlasMessage` and its subclasses (i.e. `IK` and `IC`) have access on the same `ipodlasMessageTable` instance.

The information exchange between the different instances of `IK` and `IC` deployed on the different subsystems of the IPODLAS system is based on the data structure `IM`. `IMs` are encoded in XML (cf. Figure 7-7), since XML is a simple text-based encoding offering a hierarchical structure with nested entities facilitating the parsing of datasets: For identifying a complete `IM` all text in between the tags `<IM>` respectively `</IM>` enclosing an `IM` must be parsed. When generated each `IM` gets a unique value assigned by

the associated IK instance. This value acts as unique identifier of this IM. It is held in the element tagged with <id>; currently this unique identifier is realized using an integer. Since the IMs are stored in the class attribute ipodlasMessageTable of the class IpodlasMessage the different instances of IK and IC can keep track of all existing IMs. The element <sender> represents the sender of the IM, defined sender values are listed in Table 7-4. In <rel_id> a list of the unique identifier of related IMs is held; these are the identifiers of the triggering IM(s). The element <type> can adopt the value request if the IM is a request or receipt if the IM is the receipt indicating that the an instance of IK or IC has received the request. In this case <rel_id> exhibits the identifier of the triggering request. In <cmd> the request is specified; the defined requests are listed in Table 7-5.

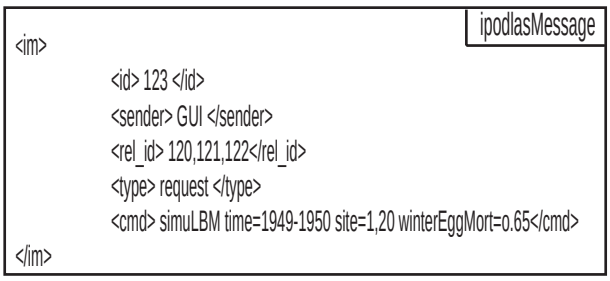

Figure 7-7 An ipodlasMessage (IM) encoded in XML. This IM has the identifier 123, it is sent from the GUI, it is related to the IMs with the identifier 120, 121, and 122, it is of type request, and it requires the IPODLAS system to simulate LBM dynamics of the years 1940 to 1950 using 0.65 as value of the simulation parameter winterEggMortality.

Defined <sender> values	Description
IK_IC_GUI	IpodlasKernel (IK) instance communicating with IpodlasClient (IC) on the GUI subsystem
IK_IC_TSS	IpodlasKernel (IK) instance communicating with IpodlasClient (IC) on the TSS subsystem
IK_IC_VR	IpodlasKernel (IK) instance communicating with IpodlasClient (IC) on the VR subsystem
IK_IC_GIS	IpodlasKernel (IK) instance communicating with IpodlasClient (IC) on the GIS subsystem
IC_GUI	IpodlasClient (IC) on the GUI subsystem
IC_ TSS	IpodlasClient (IC) on the TSS subsystem
IC_ VR	IpodlasClient (IC) on the VR subsystem
IC_ GIS	IpodlasClient (IC) on the GIS subsystem
SSC_GUI	Subsystemclient (SSC) on the GUI subsystem
SSC_ TSS	Subsystemclient (SSC)) on the TSS subsystem
SSC_ VR	Subsystemclient (SSC) on the VR subsystem
SSC_ GIS	Subsystemclient (SSC) on the GIS subsystem

Table 7-4 Defined values of element <sender> of the data structure ipodlasMessage (IM) applied in the use case 'LE3 ext'. Subsystemclients (SSC) are helper applications on the respective subsystems.

Name of request in <cmd> element	Defined <cmd> values	Description of request
identify	identify	Identify the sender of this IM.
displayInGUI	displayInGUI [text=<text>] [name=<filename>]	Display text <text> and content of file <filename> in GUI.
simuLBM	simuLBM time=<year$_1$>-<year$_2$> site=<site_no> {,<site_no>} winterEggMort=<wem>	Simulate LBM dynamics of <year$_1$> to <year$_2$> of the sites with site number(s) <site_no> {,<site_no>} and the winter egg mortality <wem>.
visuLBM	visuLBM name=<filename> time=<year$_1$>-<year$_2$> site=<site_no> {,<site_no>}	Visualize LBM dynamics, extract values from file <filename> of <year$_1$> to <year$_2$> of the sites with site number(s) <site_no> {,<site_no>}.
readControlFile	readControlFile	Read the control file on the subsystem of sender.
readDataFile	readDataFile name=<filename>	Read the data file <filename> on the subsystem of the sender.
simuWLF	simuWLF time=<year> moistureLive=<default | accLbmDef> ignitionPoint=<default | setInVr>	Simulate WLF for the year <year> using the values of the parameters moistureLive and ignitionPoint. accLbmDef means that the moistureLive value is calculated according to the current LBM defoliation, setInVr means that the ignitionPoint is selected in VR.
visuDefolLBMAndGetCoor	visuDefolLBMAndGetCoor name=<filename> time=<year>	Visualize defoliation of the year <year> extracted from the file <filename> and wait for coordinate entry of user.
simuWLF_withCoors	simWLF_withCoors time=<year> moistureLive=<default | accLbmDef> ignitionPoint=(<Easting>,<Northing>,<Elevation>) name=<filename>	Simulate WLF for the year <year> using the value of the parameter moistureLive and as ignitionPoint (<Easting>,<Northing>,<Elevation>). If moistureLive is accLbmDef the LBM defoliation is extracted from name <filename>.
visuWLFSimu	visuWLFSimu name=<filename>	Visualize WLF, read data from the file <filename>.
writtenDataFile	writtenDataFile name=<filename>	The datafile <filename> is written on subsystem of the sender.

Table 7-5 Defined values of the element <cmd> of the data structure ipodlasMessage (IM) applied in the use case 'LE3 ext' in Backus-Naur form (Naur and Backus, 1962).

The iData

Similar to IMs, the data structure iData is provided by the class IpodlasMessage. The data structure iData is applied to exchange data between the instances of IK and IC and/or the class IpodlasStorage. Each iData file is enclosed between the tags <idat> and </idat>. The first three elements form the header of the iData file: the element <name> comprises the name of the data file, in <contDesc> metainformation describing the context of the file are stored (the defined values are listed in Table 7-6), and similar as in the IM the element <rel_id> shows the identifiers of the related IM(s). The actual LBM data representing LBM migration between the sites is held in the element <data>.

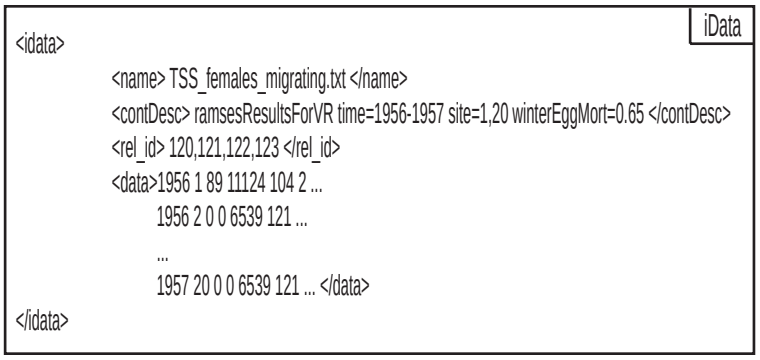

Figure 7-8 An `iData` data structure encoded in XML. The `iData` file has the name `'TSS_females_migrating.txt'`. It contains results of a simulation of RAMSES (Fischlin, 1991), and it is related to `IMs 120` to `123`. The actual LBM data is contained in the element `<data>`.

Defined `<contDesc>` values	**Description**
ramsesResultsForVR time=<year$_1$>-<year$_2$> site=<site_no> {,<site_no>} winterEggMort=<wem>	The `iData` file comprises in the `<data>` element results of a RAMSES simulation of <year$_1$> to <year$_2$> of the site(s) with site number(s) <site_no> {,<site_no>} and the winter egg mortality <wem>
lbmDefol time=<year>	The `iData` file comprises in the `<data>` element defoliation values resulting of a simulation run in RAMSES of the year <year>
wlfSpread time=<year>	The `iData` file comprises in the `<data>` element an ASCII raster representing results of a WLF simulation of the year <year>

Table 7-6 Defined values of the element `<contDesc>` of the data structure `iData` applied in the use case 'LE3 ext'.

The IpodlasStorage

The subsystem IPODLAS storage (cf. Figure 7-4) is implemented by the class `IpodlasStorage` in the IPODLAS system. The `IpodlasStorage` represents a central storage which can be accessed by all subsystems of the IPODLAS system through the kernel. It is used to store data that may be of interest for all subsystems. For example, in the final IPODLAS system simulation results generated in the TSS legacy system are stored as `iData` files in the `IpodlasStorage`. An associated metadata data structure `simMetadata` is implemented as hash table storing for each simulation result the simulation parameters used (cf. Table 7-7). When the user requests a simulation, the `IK` queries `simMetadata` whether an entry with the user-requested values of the simulation parameters already exists. If so, the associated `iData` file name is queried in `simMetadata`. Using this file name the requested file can be retrieved from the `IpodlasStorage`.

	Simulation parameters			
iData file name	Begin simulation period	End simulation period	Winter egg mortality	…
LBM_sim_1	1949	1959	0.5728	…
LBM_sim_2	1953	1958	0.65	…
LBM_sim_4	1970	1977	0.6	…
…	….	…	…	…

Table 7-7 Tabular representation of metadata structure `simMetadata`. For each key, which is the `iData` file name, several values are associated, i.e. for the key 'LBM_sim_1' the '1949', '1959', and '0.6' are associated for the simulation parameters 'Begin simulation period', 'End simulation period, and 'Winter Egg mortality', respectively.

7.3 Use case LE3 ext

To demonstrate the mode of operation of the final IPODLAS system, the scenario outlined in the use case 'LE3 ext' is described in this section. As specified in subsection 5.2.1, the use case 'LE3 ext' combines the calculation and visualization of both, Larch Bud Moth (LBM) dynamcis and Wildland fire (WLF) spread: a user simulates and visualizes LBM spread with different parameters and then starts the simulation and visualization of a WLF. The description of the sequence of actions between the subsystems of the use case 'LE3 ext' is split according to their chronology in the three subsections 7.3.1, 7.3.2, and 7.3.3, respectively.

The final IPODLAS system applies the `ipodlasMessages` (`IM`) and `iDatas` specified in Table 7-3. The system generates for all `IMs` of the type `request` transmitted between the `ICs` and `IKs` an associated IM of the type `receipt` and sends the receipt back to the sender of the `IM`. Since this action of sending a receipt takes place for each request, it is not described in the following subsections.

In final IPODLAS system, which implements the use case 'LE3 ext', 'Remote Ramses' (Bergamin, 2004) wrapping RAMSES (Fischlin, 1982), VTP[5], and GRASS (Neteler and Mitasova, 2002) are used as TSS, VR, and GIS legacy system, respectively. Since not all functionality of 'Remote Ramses' was available in the time of the implementation of final IPODLAS system, only the interfaces from and to 'Remote Ramses' were applied. So, while the IPODLAS system was designed considering 'Remote Ramses' and thus RAMSES (Fischlin, 1982), the implementation and statistical parts of the final IPODLAS system cover only the use case 'LE3 ext' as far as to the interfaces to and from 'Remote Ramses'. Since in this implementation of the use case 'LE3 ext' the prototype comprise only data from the Upper Engadine, the appropriate DEM and satellite image are already loaded into the VR subsystem (in contrast to the use case 'LE3 ext' described in subsection 5.2 and subsection 5.3). Consequently, the VR subsystem does not have to query the GIS subsystems for this data. Except for Figure 7-19 all Figures in sections 7.3 and 7.4 are achieved applying the implementation of the use cases in the prototype IPODLAS system.

7.3.1 LBM simulation and visualization

When starting the IPODLAS system the `IpodlasClient` (`IC`) instances on the participating subsystems GUI, TSS, VR, and GIS request socket connections to the `IpodlasKernel` (`IK`). For each connection an instance of `IK` is generated, which receives the `IM identify`, identifying the respective `IC`.

[5] http://www.vterrain.org/ (accessed January 11, 2006)

GUI to Kernel

Figure 7-9 illustrates the starting point of the use case 'LE3 ext' showing the LBM configuration widgets of the IPODLAS GUI. In the window 'LBM configuration' the user selects that she/he wants to see the simulation output in 3-D and that the simulation model applying the 'Food quality hypothesis' (Fischlin, 1982; Fischlin and Baltensweiler, 1979) and LBM migration (Fischlin, 1982) is to be used (cf. subsections 5.1.1 and 5.1.4). The window 'Food quality and migration' is used to select the values for the time period of LBM simulation; for all other simulation parameters the default values are taken. As described in the 'GUI2VR' prototype (cf. subsection 6.2.4), after pressing the 'Run' button, the values entered in the GUI are catched by AppleScripts. They invoke processes providing helper functionality (inter alia SSC_GUI) to generate the respective IM `simuLBM` and to write it to a local file, the controlFile (cf. Figure 7-10). The IM `readControlFile` is used to invoke the `IC` on the GUI subsystem to read the control file. It parses the `IM simuLBM` and sends it through the pre-generated socket connection to the associated instance of the `IK` on the kernel subsystem.

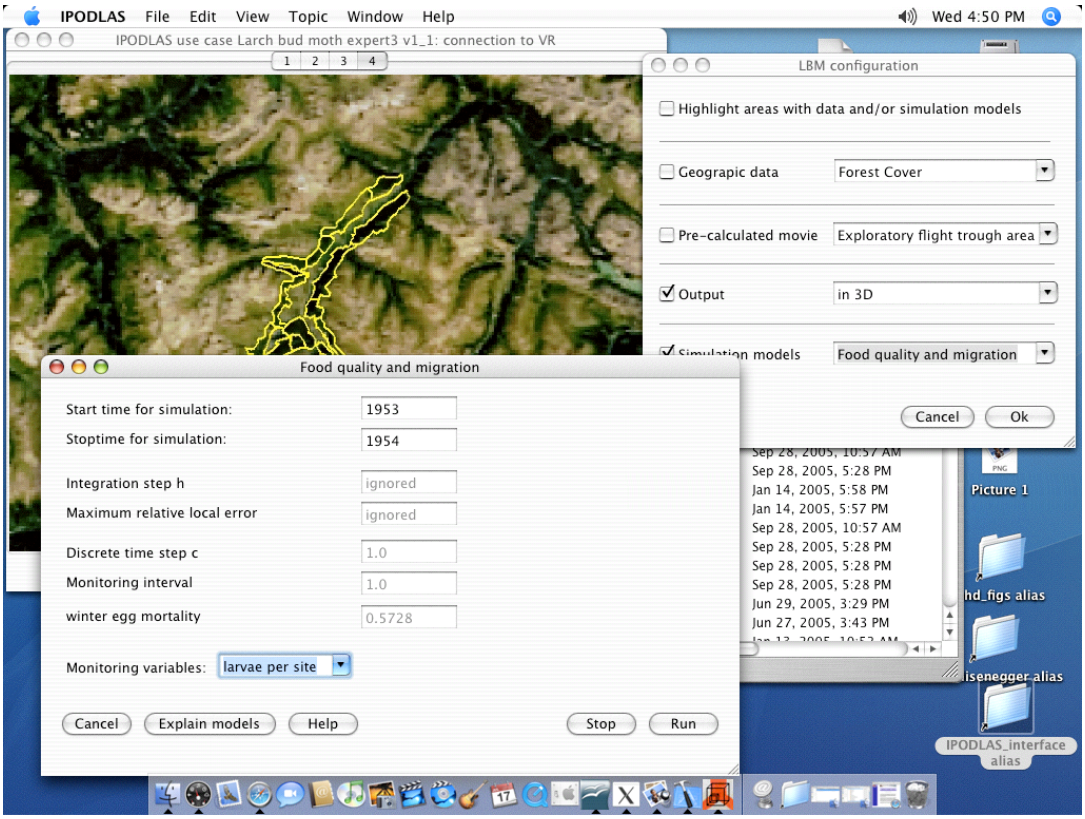

Figure 7-9 Widgets of the IPODLAS GUI to configure an LBM simulation applied in the use case 'LE3 ext'. In the window 'LBM configuration' the user selects the output mode of the simulation and the simulation model to be used. The values for the simulation parameters can be determined in the window 'Food quality and migration'. The IPODLAS GUI has been developed by the author in the IPODLAS approach (cf. chapter 5) to support the specification of the use cases. It is (re)used and extended here in the implementation of the use case 'LE3 ext'.

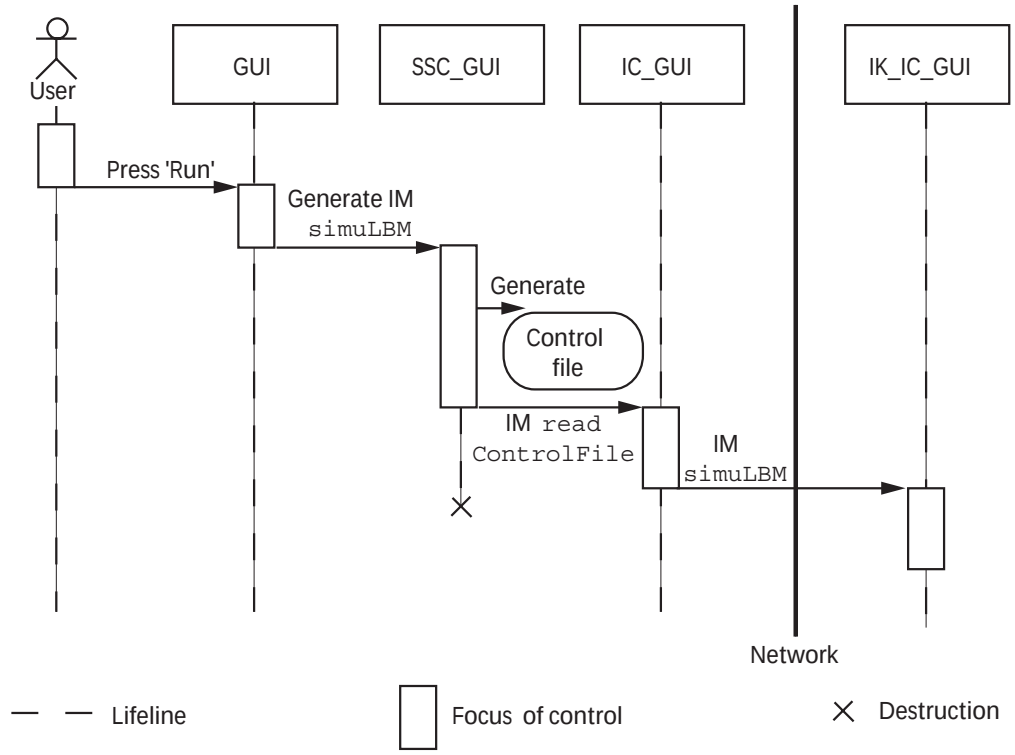

Figure 7-10 Sequence diagram[6] of the use case part described in the paragraph 'GUI to Kernel'. The diagram shows the flow of control between the participating processes (`GUI`, `SSC_GUI`, `IC_GUI`, `IK_IC_GUI`) in this part of the use case. Names of defined participating processes for the use case 'LE3 ext' are listed in Table 7-4. `SSC_GUI` stands for a number of helper applications, which exist only during a limited time span. The lifeline illustrates the period within which the respective process exists; the focus of control indicates which process is in control at which period. Transmitting `IMs` and `iDatas` between different subsystems over socket connections is indicated by the black line symbolizing the network.

Kernel to TSS

The responsible `IK` instance generates an entry with a unique identifier in the class variable `ipodlasMessageTable` of the class `IpodlasMessage`. Figure 7-11 shows that the `IK` checks if the `IpodlasStorage` contains the results of the requested LBM simulation. If not, the `IM simuLBM` is handed over to the `IK` instance associated with the `IC` on the TSS subsystem to execute the request specifed in the `IM`. This `IK` instance sends the `IM simuLBM` to the `IC` on the TSS subsystem. The `IC` triggers the XML-RPC-Client of 'Remote Ramses' (Bergamin, 2004) (the interface to the TSS legacy system, cf. subsection 6.2.3) to start a LBM simulation using the values for the time period read from the transmitted `IM simuLBM`. When the simulation has finished 'RemoteRamses' writes the simulation results to a local data file. The `IC` is invoked by the helper application `SSC_TSS` through the `IM readDataFile` to read the resulting simulation output from the local data file. After parsing the local data file, the `iData ramsesResultsForVR` (cf. Table 7-6) containing the requested simulation information and the associated `IM visuLBM` are generated and sent via the `IC` to the associated `IK` instance.

[6] "The sequence diagram shows how the focus — starting at the upper left corner — moves from object to object as the use case is performed and messages are sent between objects. A message sent from one object triggers the receiving object to take over focus […]" (Jacobson et al., 1999, p. 51).

Kernel to storage

The responsible `IK` instance generates an entry in class variable `ipodlasMessageTable` of the class `IpodlasMessage` and stores the `iData` file in the `IpodlasStorage`. Additionally, as described in subsection 7.2.2, the `IK` instance generates an entry in the metadata storage describing the LBM result file through the parameter values used to generate it.

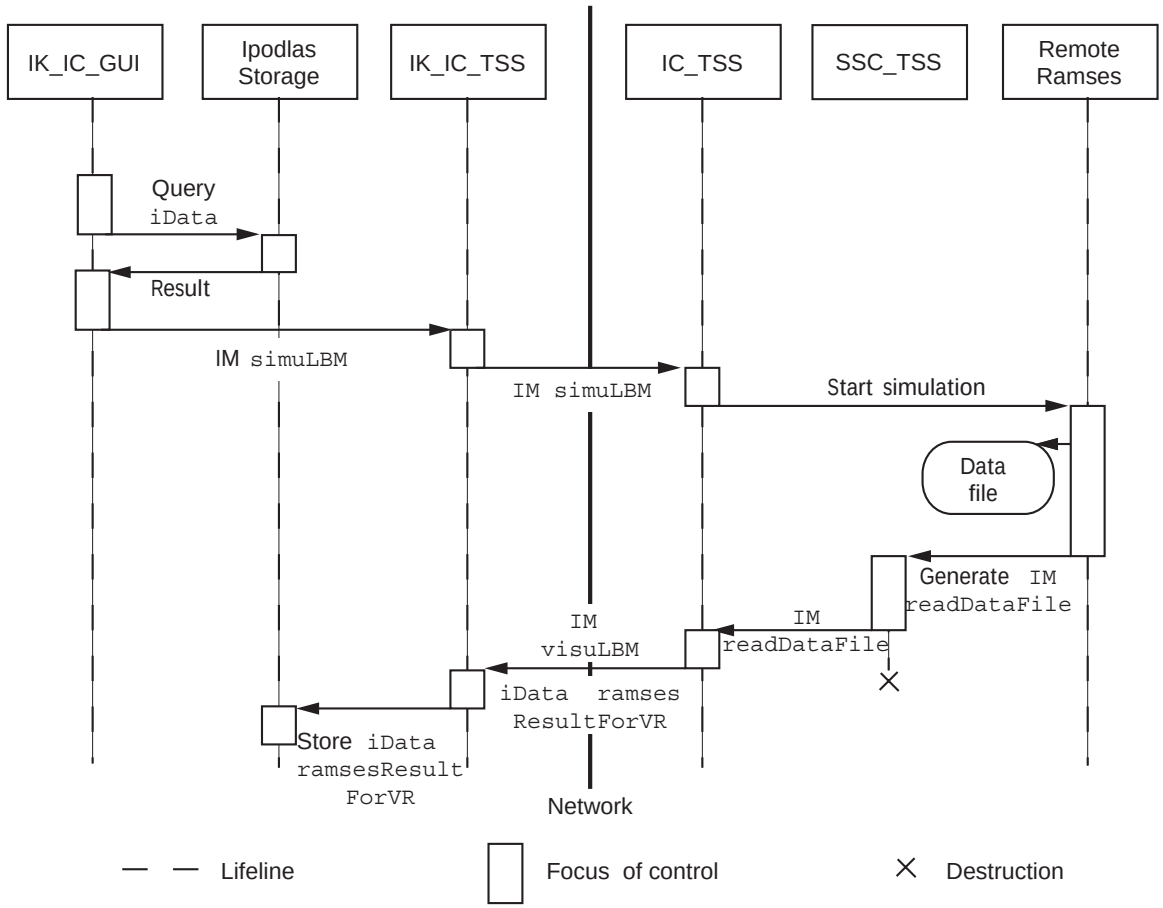

Figure 7-11 Sequence diagram of the use case part described in the paragraphs 'Kernel to TSS' and 'Kernel to storage'.

Kernel to GUI

As Figure 7-12 illustrates, at this point in the use case, two parallel threads of actions are initiated. The first thread is described in this paragraph 'Kernel to GUI' and the second thread in the next paragraph 'Kernel to VR'. The `iData ramsesResultsForVR` containing the LBM simulation results and the associated `IM displayInGUI` are transmitted by the `IK` instance associated with the GUI to the `IC` of the GUI subsystem. The `IC` writes the `iData ramsesResultsForVR` to a local data file and invokes the GUI to display the content of the local data file to the user.

Kernel to VR

In parallel to the actions described in paragraph 'Kernel to GUI', the `IK` instance associated with the VR subsystems sends the `iData ramsesResultsForVR` comprising the LBM simulation result and the associated `IM visuLBM` to the `IC` on the VR subsystem, which writes the data structure `iData` to a local data file. After doing that, the `IC` invokes the VTP[7] (the VR legacy system) to read the `iData` from the local data file and to start the visualization of the LBM result (cf. Figure 7-13). When the visualization has finished, the VR legacy system invokes the `IC` on the VR subsystem to send the `IM displayInGUI` to the associated `IK` instance. The `IM` is propagated to the `IK` instance associated with the GUI, which in turn invokes the `IC` on the GUI subsystem to tell the GUI to notify the user about the termination of the animation.

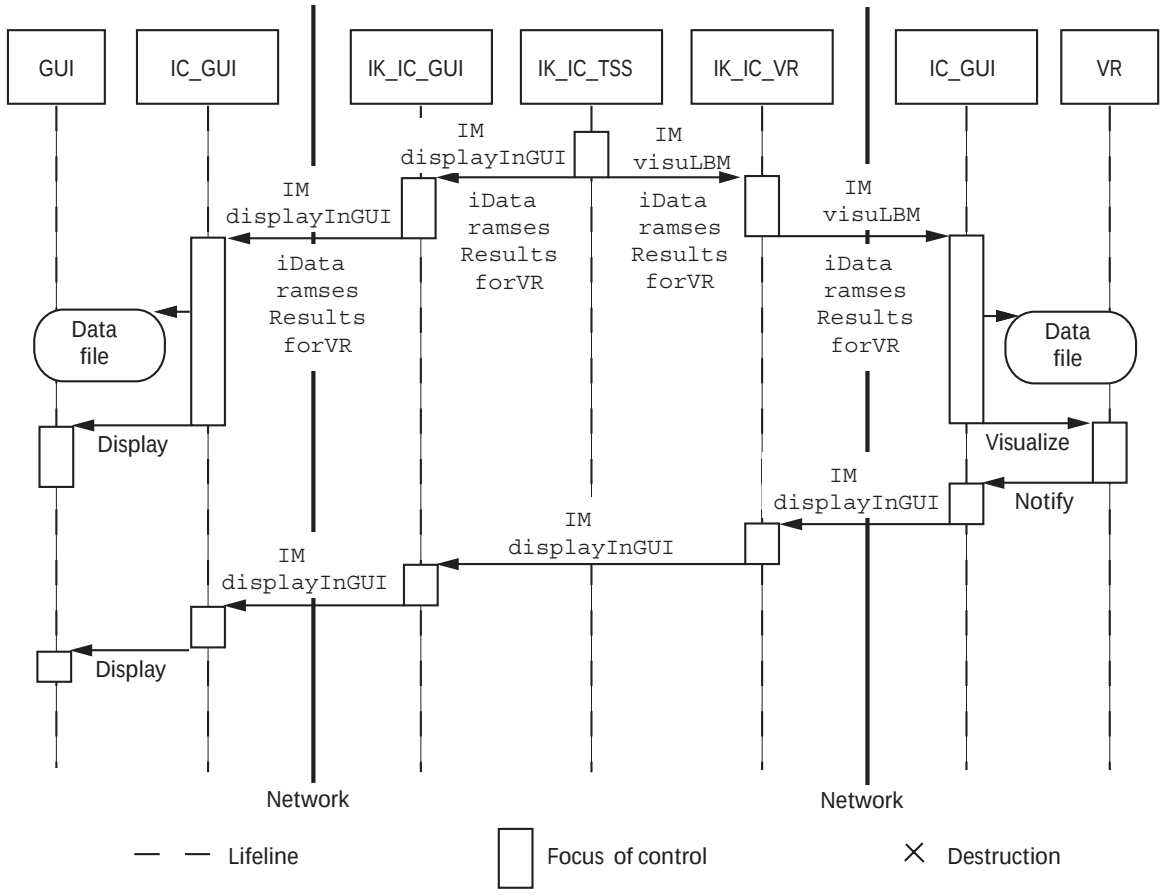

Figure 7-12 Sequence diagram of the use case part described in the paragraphs 'Kernel to GUI' and 'Kernel to VR'. In this Figure and in the following the representation of the helper applications (for which `SSC` is taken as representative) are omitted for the sake of clarity.

[7] http://www.vterrain.org/

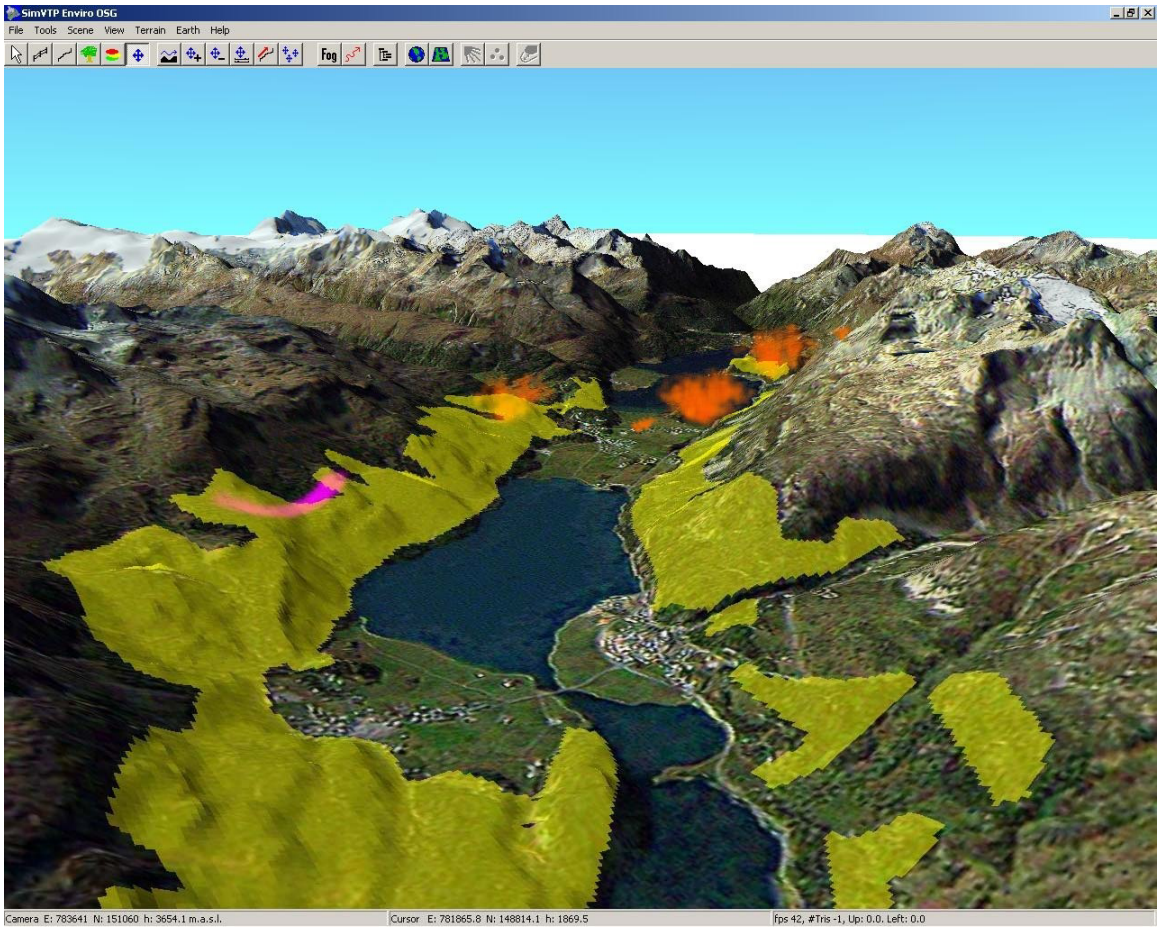

Figure 7-13 Visualization of LBM simulation results in VTP. The forests are colored according to their defoliation rate (here in a greenish yellow), the LBMs are visualized as clouds migrating from their origin site to their target site. The LBM clouds are colored according to their site they of origin (Image courtesy of Y. Wu).

7.3.2 LBM simulation and visualization with different parameters

In this second part of the use case 'LE3 ext' the user wants to compare the generated LBM simulation results where default values for the simulation parameters have been applied (the generation process is described in subsection 7.3.1) with an LBM simulation using a higher value for the simulation parameter 'Winter egg mortality' (cf. Figure 7-9). Figure 7-14 shows that this simulation follows the same sequence of actions as described in subsection 7.3.1, with one exception. If it is assumed that the user-requested simulation results have already been generated in a previous simulation run (i.e. the same values for all simulation parameters have been applied in a previous simulation run), then the invocation of the TSS subsystem is omitted. In this case, the `IK` detects in the metadata structure `simMetadata` an entry with the same values for the simulation parameters as in the user-requested simulation (cf. subsection 7.2.2). The retrieved filename associated with this entry is used to identify the `iData` file in the `IpodlasStorage`. The `IK` then reads the `iData` file and sends it to the GUI and the VR subsystem, where the LBM simulation results are displayed as described in subsection 7.3.1.

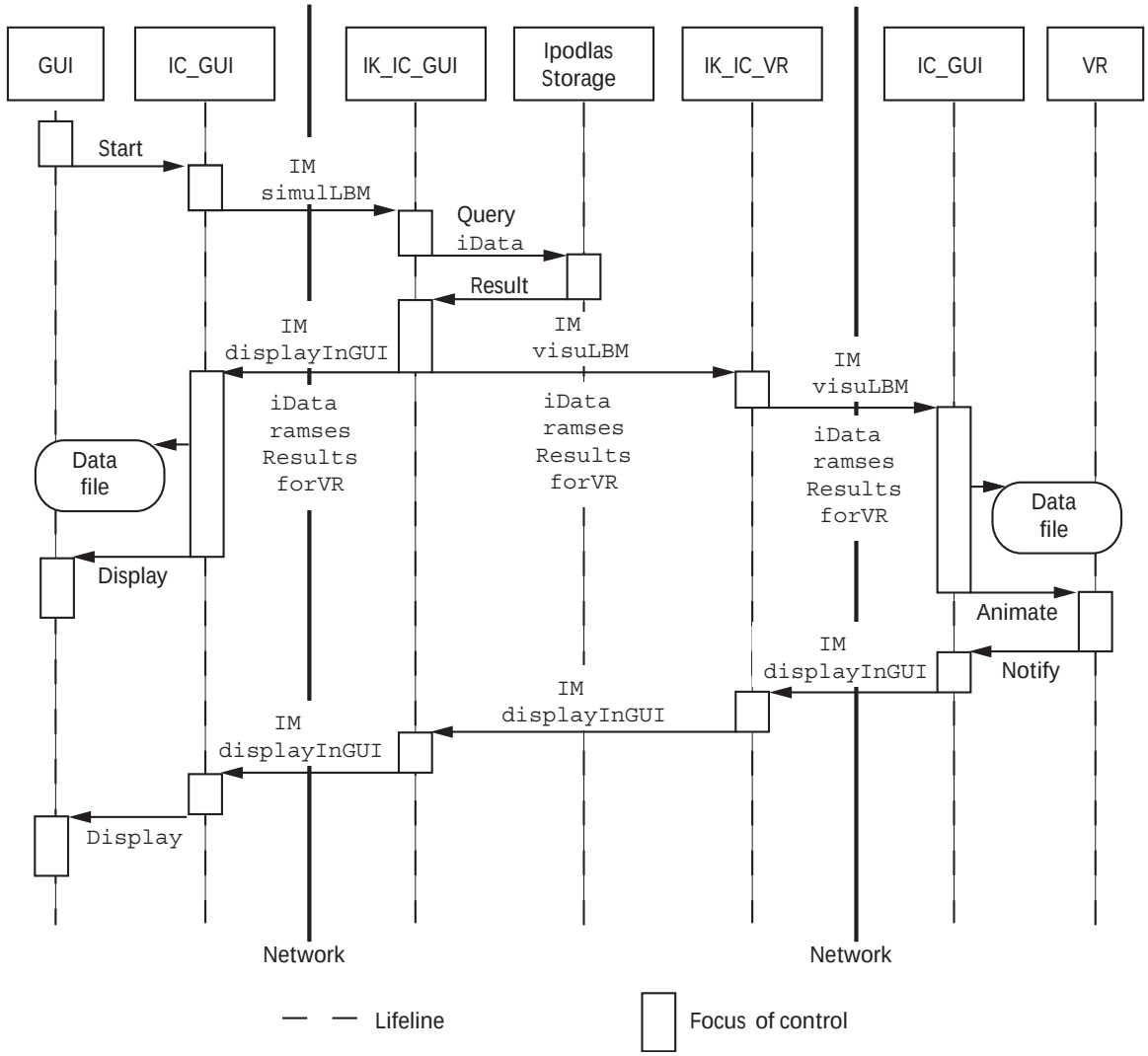

Figure 7-14 Sequence diagram of the use case part described in subsection 7.3.2. It shows a similar sequence of actions as Figure 7-10, Figure 7-11, and Figure 7-12 combined, but on a higher level of generalization and omitting the part of the use case of the TSS subsystem.

7.3.3 WLF visualization

In the third part of the use case 'LE3 ext' the user wants to examine a WLF spread in areas with trees defoliated by LBM.

GUI to kernel

In Figure 7-15 the widgets of the IPODLAS GUI for configuring WLF simulations are displayed. In the 'Wildland fire configuration' window, the user selects 3-D simulation output visualization and that the WLF spread is to be calculated in GRASS (Neteler and Mitasova, 2002) (cf. subsection 3.3.4). In the window 'Wildland fire simulation in GRASS' the user can select the begin and end of the WLF spread simulation. Assuming that the moisture of live vegetation is decreased when the vegetation, i.e. in this case the larch trees are defoliated by the LBMs, the user selects that the simulation parameter

'Moisture live'[8] is to be calculated from existing LBM simulation data. Additionally, she/he selectes that the origin of the WLF spread (simulation parameter 'Ignition point') is to be set by the user in the VR legacy system. Applying the same mechanisms as described in subsection 7.3.1 the `IM simuWLF` is generated and sent from the `IC` on the GUI subsystem to the associated `IK` instance.

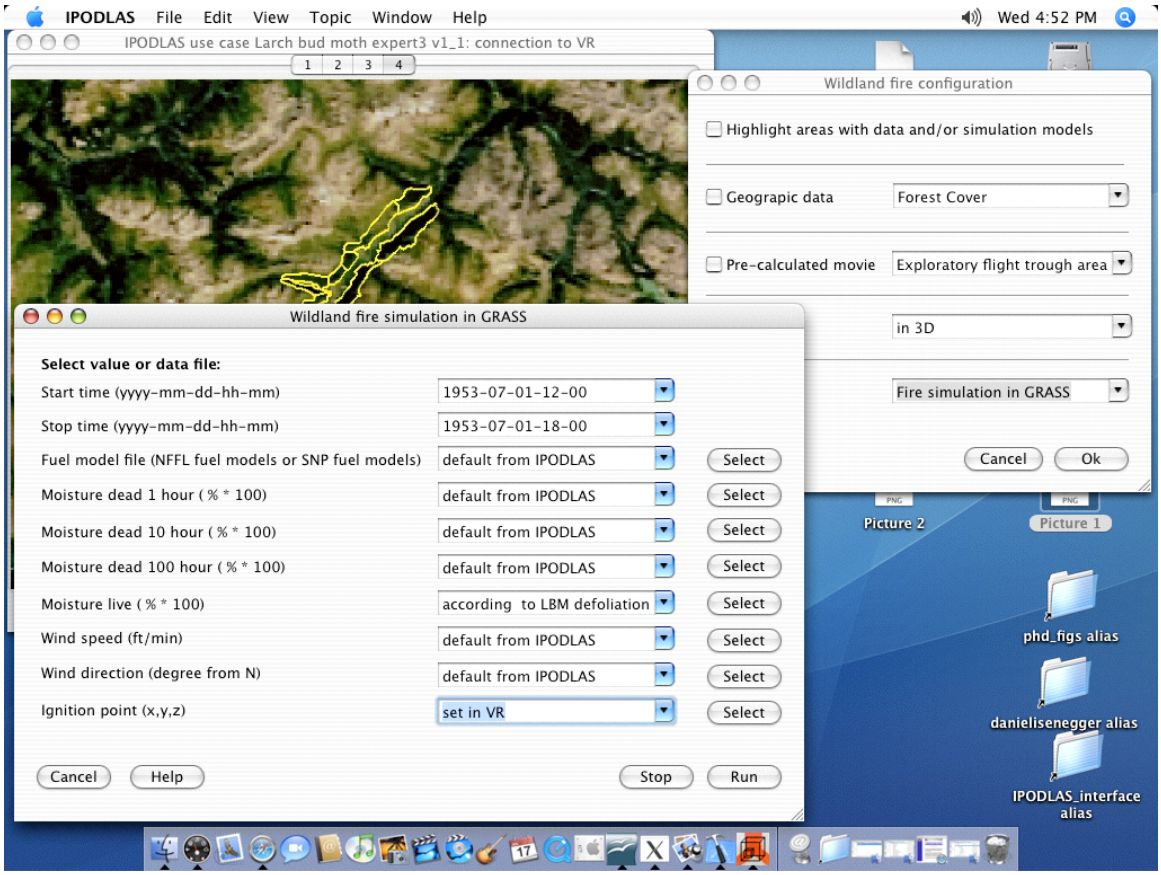

Figure 7-15 Widgets of the IPODLAS GUI to configure a WLF simulation applied in the use case 'LE3 ext'. Similar as shown in Figure 7-9 in the window 'Wildland fire configuration' the user selects the output mode and the model to be used. The values for the simulation parameters can be entered in the window 'Wildland fire simulation in GRASS'.

Kernel to VR

In Figure 7-16 it is illustrated that the responsible `IK` instance receives and parses the `IM simuWLF`. The `IK` instance then generates a new item in the class variable `ipodlasMessageTable` of the class `IpodlasMessage`. If LBM defoliation data from the user-requested year can be retrieved in the `IpodlasStorage`, the `iData lbmDefol` comprising the respective defoliation data and the associated `IM visuDefolLBMAndGetCoor` is generated. The `IC` instance on the VR subsystem gets the `iData lbmDefol` and the `IM visuDefolLBMAndGetCoor` from the `IK` instance associated with the VR subsystem. The LBM defoliation data contained in the `iData` is written to a local data file. The `IC` instance invokes the VR legacy system VTP[9] via

[8] While in the Rothermel model, in SPARKS, and in FARSITE the fuel moisture is classified according to its size (diameter) in 3 classes for the dead and in 2 classes for the live fuel, the applied WLF simulation model r.spread only uses one live fuel moisture class.

[9] http://www.vterrain.org/ (accessed January 11, 2006)

socket connection to read the local data file and to visualize the defoliation rate of the forest. In the VR legacy system the defoliation rate is symbolized by different colors; the user then can select the origin of a WLF (most likely in a highly defoliated forest area) using a pointing device (cf. Figure 7-17). The `IC` on the VR subsystem receives the coordinates of the WLF origin from the VR legacy system, generates the IM `simuWLF_withCoors` and sends this IM to the associated IK instance.

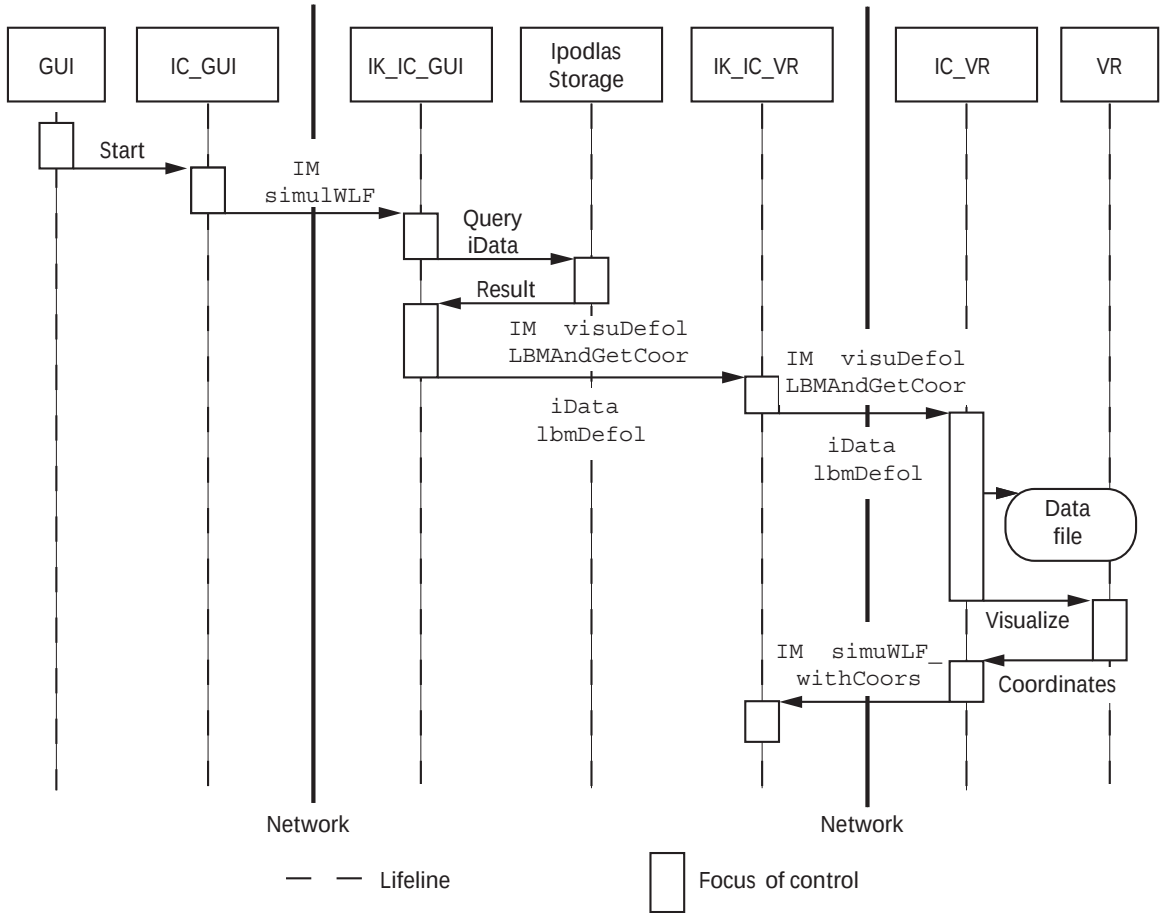

Figure 7-16 Sequence diagram of the use case part described in the paragraphs ‘GUI to kernel’ and ‘Kernel to VR’.

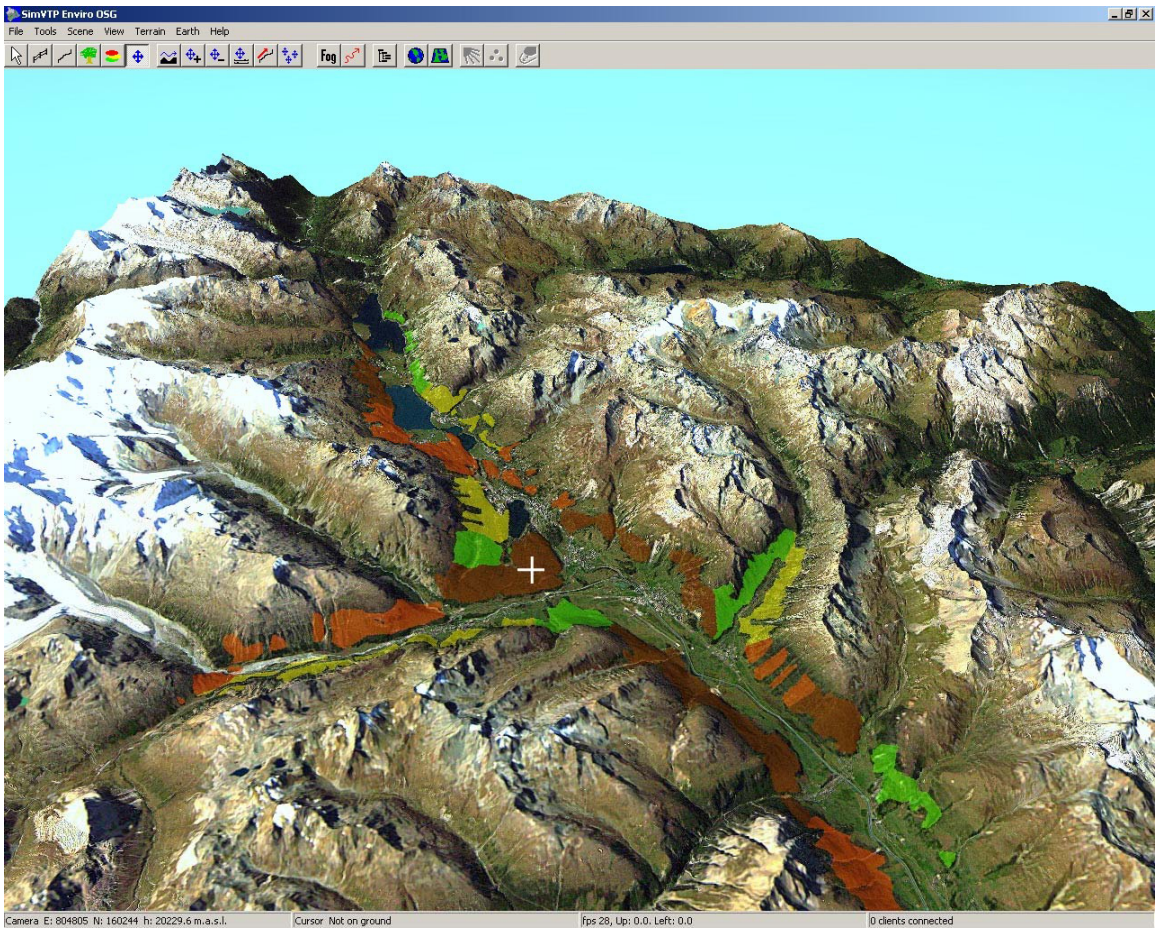

Figure 7-17 Visualization of the defoliation rates of the sites in the Upper Engadine: from green meaning low defoliation rates over yellow and orange to brown representing high defoliation rates. In the use case 'LE3 ext' the user can select in a site — probably in one exhibiting a high defoliation rate — the origin of the WLF using a pointing device; symbolized here by the white cross (Image courtesy of Y. Wu).

Kernel to GIS

The `IK` instance associated with the GIS subsystem gets and parses the `IM simuWLF_withCoors` (cf. Figure 7-18). The `IK` instance queries the required LBM data and sends the LBM data as `iData lbmDefol` together with the associated `IM simuWLF_withCoors` to the `IC` on the GIS subsystem. This `IC` instance invokes the GIS legacy system GRASS (Neteler and Mitasova, 2002) to calculate a WLF spread. GRASS calculates the live fuel input data by decreasing the default live fuel values by a factor according the amount of defoliation calculated by the TSS subsystem for the corresponding area: the higher the defoliation the larger the decrease of the default live fuel values. GRASS then starts the WLF simulation *r.spread* (Xu, 1994) using the coordinates of the WLF origin (selected by the user in VTP) which have been transmitted in the `IM simuWLF_withCoors`. The result of the WLF simulation is an ASCII-raster whose values symbolize the time when the WLF arrives at the individual cell. The raster is wrapped in the `data` element of the `iData` structure to the `iData wlfSpread` and transferred from the `IC` on the GIS subsystem to the associated `IK`. The `IC` on the VR subsystem is invoked from the responsible `IK` instance through the `IM visuWLFSimu` to write the received `iData wlfSpread` to a local data file, then the IC invokes VTP to visualize the WLF spread (cf. Figure 7-19.

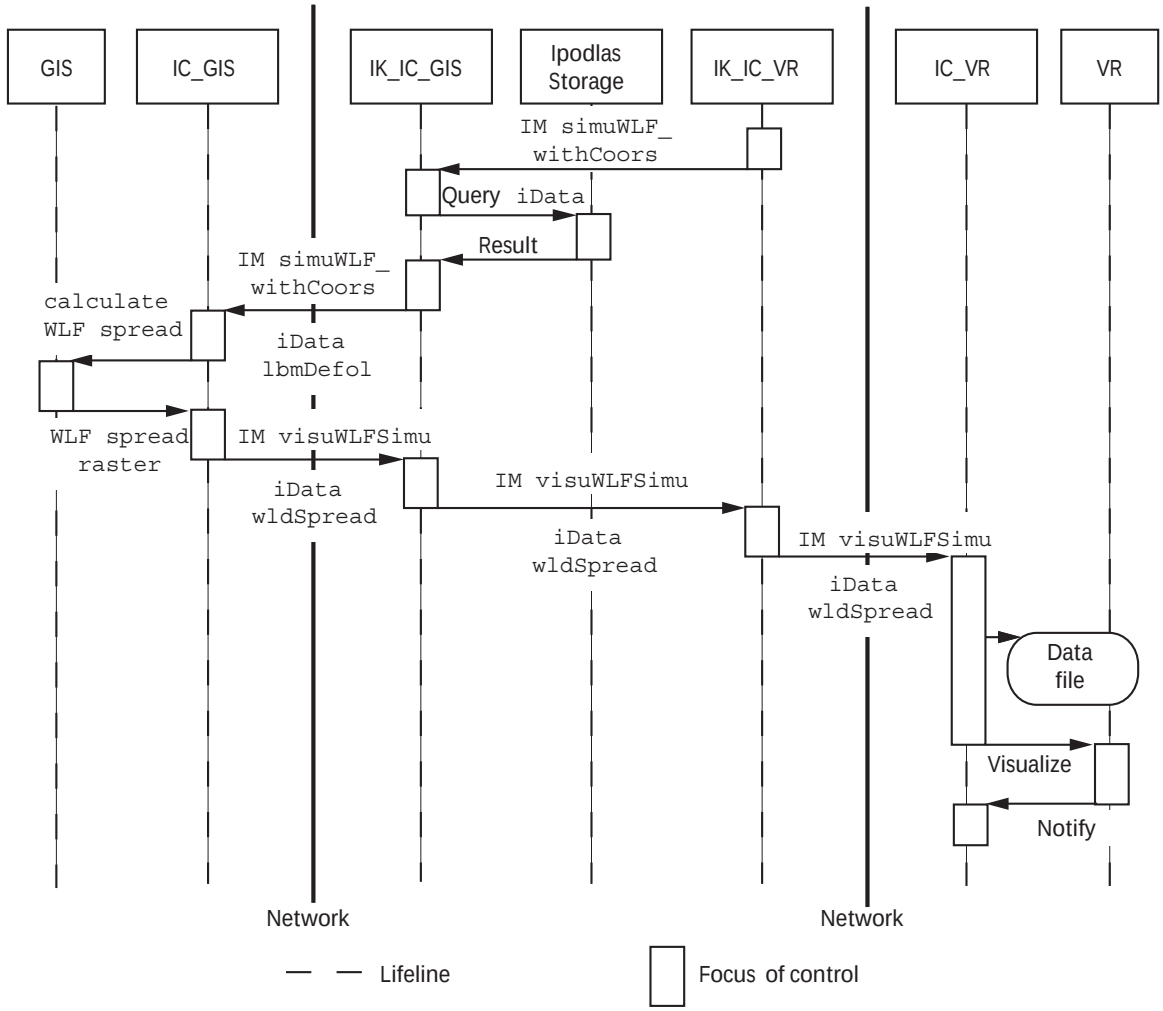

Figure 7-18 Sequence diagram of the use case part described in the paragraph ‘Kernel to GIS’.

Figure 7-19 Visualization of the WLF spread in the VR legacy system VTP. In the use case 'LE3 ext' VTP visualizes the raster `wlfSpread` representing a WLF spread, which has been calculated in the GIS subsystem and transferred to VTP as `iData wlfSpread`. The WLF exhibits the form of an ellipse elongated uphill and in wind direction (Image courtesy of Y. Wu. This picture has not been generated using the IPODLAS system; it is shown here, since it neatly illustrates VTP's wildland fire visualization capabilities).

7.4 Performance measurements in the use case 'LE3 ext'

The sequence diagrams in section 7.3 illustrate that in the use case 'LE3 ext' a lot of data, i.e. `ipodlasMessages` (`IM`) and `iDatas`, are exchanged. For upscaling and extending the IPODLAS system (e.g. adding more subsystems or adding new functionality) it is important to be able to estimate its performance. Firstly, the transfer times of `IMs` and `iDatas` between the subsystems have to be measured and secondly how the run time is distributed among the different parts of the IPODLAS system. In subsection 7.4.1 the transfer times of `IMs` and `iData` applied in the use case 'LE3 ext' are analyzed. The run times of the individual parts of the IPODLAS system applying 'LE3 ext' are examined in subsection 7.4.2.

The performance measurements were accomplished with the following deployment of the IPODLAS system:

- the GIS subsystem, the `IK`, and the `IpodlasStorage` on a Pentium III Coppermine (864 MHz, 768 MB RAM) running on SUSE 9.1.
- the GUI and the TSS subsystem on a Power PC G4 (733 MHz, 768 MB SDRAM) using Mac OS 10.3.9.
- the VR subsystem on a Pentium 4 (2.4 GHz, 1.023 GB RAM) running on Windows XP.

The subsystems are connected through a 100Mbit/s local area network (LAN).

The time measurements in the use case 'LE3 ext' have not been conducted for exact quantitative statements, but primarily for comparative estimation of transfer times of `IMs` and `iDatas` and of proportions of time usage of the subsystems involved in 'LE3 ext'.

7.4.1 Transfer time of ipodlasMessages (IM) and iData

In the following the transfer time of IM or iData denotes the amount of time that is needed to send an IM or an iData from an IK to an IC or vice versa. The transfer time is measured by taking the time of the system clock when the line of code is executed where the data is received minus the time of the system clock when the line of code is executed where the data is sent. In Table 7-8 the transfer times of IMs and iDatas used in the use case 'LE3 ext' are listed. The transfer time measurements were collected in nineteen runs of 'LE3 ext'. The number of transfer time measurements listed in Table 7-8 for a given IM or iData cannot be directly compared with the number of occurence of the respective IM or iData in 'LE3 ext', since not in each test run the whole use case 'LE3 ext' has been conducted or evaluated. Due to problems in synchronizing the system clock of the VR subsystem with the system clocks of the other subsystems, for IMs and iDatas from and to the VR subsystem not the one-way transfer time, but the round-trip transfer time has been recorded. Round-trip transfer time means that, for instance, the transfer time of the IM simWLF_withCoors from the IK to the IC on the VR subsystem and the transfer time of the IM visuDefolLbmAndGetCoors from the IC on the VR subsystem to the IK is added for measurement. In order to to compare the run time properties of the round-trip measurements with the properties of one-way measurements, the transfer times and the volume of transferred data are averaged.

Transferred IM or iData and size	**Transfer time [s]**									**Average [s]**	**Std-dev**
IM readControlFile (175 bytes)	0.012	0.001	0.011	0.005	0.012	0.016				0.010	0.005
IM simWLF_withCoors + IM visuDefolLbmAndGetCoor (220 bytes)	0.016	0.005	0.015	0.011	0.007	0.008	0.014			0.010	0.005
IM simuLBM (225 bytes)	0.005	0.004	0.012							0.007	0.004
IM simWLF_withCoors (260 bytes)	0.001	0.001	0.002	0.002	0.002	0.004	0.001	0.005	0.001	0.002	0.001
iData ramsesResultfile + IM writtenDataFile (4000 bytes)	0.009	0.002	0.005	0.008	0.004	0.009	0.017			0.007	0.005
iData wlfSpread (9000 bytes)	0.002	0.002	0.002	0.002	0.001	0.002	0.012			0.003	0.004

Table 7-8 Measured transfer times of ipodlasMessages (IM) and iDatas applied in use case 'LE3 ext'. Rows involving only one IM or iData list the one-way transfer time. In rows with two IMs or with one IM and one iData half the round-trip transfer time and half the sum of the data size is listed. The sizes of the IMs and iDatas in bytes are rounded average sizes[10].

Figure 7-20 depicts the average transfer times of the IMs and iDatas listed in Table 7-8. Altough the significance of this data is limited due to the small amount of measurements, the distribution of the measured transfer times indicates that there is no direct relation

[10] The size of the IMs and iDatas can be variable due to the variable nature of XML: the values of the elements can adopt arbitrary texts, affecting the data size.

between the amount of the transferred data and the time required to transfer the data. For example, the average transfer time of the `IM readControlFile` (220 Bytes) is larger than the average transfer time of the `iData wlfSpread` (9000 Bytes) (cf. Table 7-8). Furthermore, the transfer time of all `IMs` and all `iDatas` exhibit comparatively large standard deviation. This suggests that other factors, for instance CPU usage by other processes or network delays, are the main factors influencing the transfer time rather than the amount of data transferred.

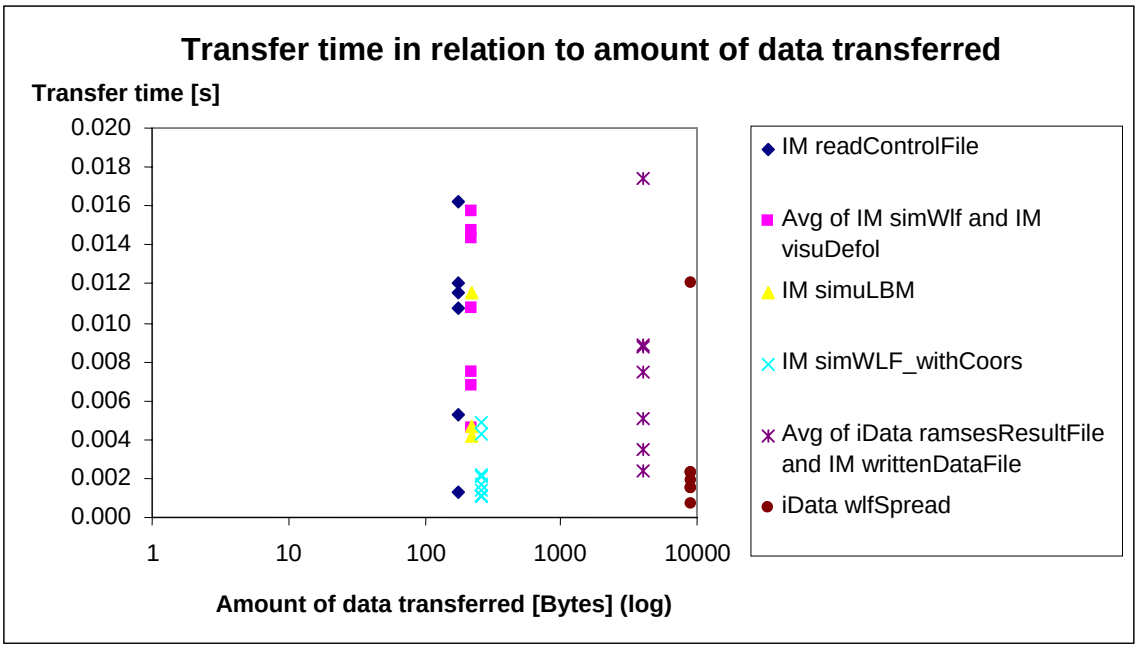

Figure 7-20 Transfer times of `IMs` and `iData` in relation to logarithm of the amount of data transferred. For the `IMs simWlf` and `visuDefol` respectively the `iData ramsesResultFile` and the `IM writtenDataFile` half the round-trip transfer and half the sum of the data size is displayed. For all other `IMs` or `iDatas` the one-way transfer time is represented.

7.4.2 Run time of the final IPODLAS prototype

For advancing the IPODLAS system, it is interesting to know the contribution of the individual parts of the IPODLAS system to the run time. In this subsection the average run time occupied by the three parties *user*, *legacy systems*, and the *rest of the IPODLAS system* is examined. Table 7-9 lists the run time of the final IPODLAS system executing the use case 'LE3 ext', where simulation and visualization of 2 years of LBM migration are requested. In Table 7-10 the same listing scheme is applied on the use case 'LE3 ext' requesting simulation and visualization of a WLF.

In both run time measurements the proportions of the values to each other may be compared, not the absolute values, since most measurements are dependent on diverse variable constraints (e.g. user, the simulation parameters, etc.). However, the overall properties are evident. The IPODLAS system (without the legacy systems) consumes about one percent of the total run time of the use case 'LE 3 ext'; the rest is consumed by the user and the legacy systems. Thus, there seems to be enough potential for extending the IPODLAS system. The run time measurements indicate that even a considerable increase of network traffic within the IPODLAS system is not likely to change the run time distribution significantly.

Task	**Time [s]**	
Average user-GUI interaction time to configure and start (user-dependent[11])	30.00	
Average run time to process user interaction and propagate request to kernel	0.28	
Average run time to process a LBM simulation request and to propagate request from kernel to TSS and back	0.50	
Average run time to process actions and to propagate request from kernel to VR	0.15	
Average visualization time in VR of a LBM migration simulation of 2 years (user-dependent)	45.00	
Total	75.93	**%**
Time spent in the IPODLAS system, excluding the legacy systems	0.93	1.22
Time spent by the user	30.00	39.51
Time spent by visualization	45.00	59.27

Table 7-9 Run times of the IPODLAS system conducting within the use case ‘LE3 ext’ a request for simulation and visualization of a LBM migration simulation of 2 years. The user interaction time and the visualization time is strongly dependent on the user.

Task	**Time [s]**	
Average user-GUI interaction time to configure and start (user-dependent)	30.00	
Average run time to process user interaction and propagate request to kernel	0.28	
Average run time to process the request and propagate request from kernel to VR	0.15	
Average user-VR interaction time to select a WLF ignition point in VR (user-dependent)	10.00	
Average run time to process a WLF simulation request and to propagate request from kernel to GIS and VR	0.50	
Average WLF simulation time in GRASS	165.50	
Average WLF animation time in VR (user-dependent)	45.00	
Total	246.43	**%**
Time spent in the IPODLAS system, excluding the legacy systems	0.93	0.38
Time spent by the user	30.00	12.17
Time spent by the GIS	165.50	67.16
Time spent by visualization	40.00	16.23

Table 7-10 Run times of the IPODLAS system conducting within the use case ‘LE3 ext’ a request for simulation and visualization of an WLF spread. While the user interaction time and the visualization time is dependent on the user, the WLF simulation time has been estimated by taking default values for the WLF simulation in GRASS.

[11] User-dependent means that the time the user spends to interact with the system is arbitrary, depending e.g. on the users experience or the visualization preferences.

Part III

8 Discussion

'Part III' encapsulates the evaluating and concluding parts — the discussion and the conclusions — of the thesis. This chapter addresses the research questions of this thesis specified in chapter 1 and offers answers by providing a synthesis of the chapters 5, 6, and 7. Table 8-1 recapitulates the research questions and specifies which research question is discussed in which section of this chapter. The discussion is structured according to the partition of 'Part II' of the thesis. Each chapter of 'Part II' is addressed in an individual section of the discussion.

Research questions		**Addressed in chapter of the thesis**	**Addressed in section of the Discussion**
1.	What is the appropriate development methodology to gather the full range of user requirements and system constraints for the development of a system such as IPODLAS?	5, 6	8.1, 8.2
2.	Is the standard GIS functionality sufficient to support the requirements of a system such as IPODLAS?	5	8.1
3.	What are suitable concepts and architectures for a software system to meet the goals of the IPODLAS project, which are to develop a system combining the three domains, GIS, VR, and TSS to facilitate the joint seamless usage of functionality, data, and models?	7	8.3

Table 8-1 The Table illustrates in which chapters of the thesis the research questions specified in chapter 1 are mainly addressed respectively in which section of the chapter 8 the research questions are mainly discussed.

8.1 The IPODLAS approach

This section discusses the IPODLAS approach, which is described in chapter 5 of this thesis. Evaluating the development methodology of the IPODLAS framework and specifying the required GIS functionality the research questions 1 and 2 are addressed in the subsections 8.1.1 and 8.1.2, respectively.

8.1.1 The methodology

The concepts and methodology of the approach to develop the IPODLAS system are discussed in this subsection. Focussing on the methodological aspects of this thesis the evaluation of the development methodology IPODLAS approach addresses research question 1.

In this thesis legacy systems from different domains providing models on different scales are applied to better represent spatiotemporal, cross-scale environmental processes. Besides the development of the IPODLAS system, the author sees also the development approach — the IPODLAS approach — as a result. In this respect, this thesis makes also methodological contributions. The IPODLAS approach can be seen as a methodology to address interdisciplinary projects exhibiting multiple, partially ambiguous requirements and exhibiting a considerable software development component. To challenge the capabilities of the IPODLAS system to be able to deal with spatiotemporal and cross-scale processes *case studies* from different domains provide data from spatiotemporal and cross-scale processes and different models. To represent cross-

scale processes more appropriately the simulation model representing the process is chosen according to the observation scale. Since the models acting on different scales can be coupled, the representation of cross-scale processes is supported. This is the core concept of the case study framework (cf. Table 5-1), where each case study provides models on three different scales.

The structured approach of the 'Unified software development process' (UP) (Jacobson et al., 1999) is a formal and transparent way for developers to support the determination and description of user requirements and to move from requirements to a software system. *Use cases* are applied in the UP to systematically and intuitively capture the functional requirements of different users and to specify usage scenarios of the IPODLAS system. The graphical specification of the GUI of the IPODLAS system helps to define functionality the users can interact with at each stage of the use case and thus supports the specification of the use cases. The refinement of the use cases in the sequenced action lists allows the specification of the actions of a use case in a much finer granularity than using only the prose description. The *sequenced action lists* and the associated graphical definitions of the IPODLAS GUI define the state of the GUI and the functionality offered to the user for each relevant point in time of the use case; this is important for a user-driven system where each action of the system is triggered by a user interaction. The *use case model* consists of all use cases developed within the IPODLAS framework (cf. Table 5-3).

The sequenced action lists focus on the interactions of the user with the GUI. The *functionality listings* (cf. section 5.3) enlarge the focus to the interactions between the subsystems of the IPODLAS system. The functionality listings are based on the sequenced action lists: the listings track the cascade of actions initiated by a user interaction on the IPODLAS GUI within the IPODLAS system by specifying which part of the required functionality is provided by which subsystem. This generates on the one hand a specification of the sequence of actions which are necessary to implement the corresponding use case. On the other hand, the functionality listings specify which subsystem provides which piece of functionality. Regarding the GIS subsystem, the results of the functionality listings are used in subsection 8.1.2 to address research question 2. Furthermore, the functionality required from the subsystems is classified according to the classification scheme in Table 5-2 (cf. subsection 5.1.5). This facilitates an estimation of the implementation effort necessary to provide the required functionality within the IPODLAS system.

In this thesis where integration of legacy systems is an important part, the functionality listings are a concise and structured instrument to determine and evaluate the necessary efforts to integrate existing and lacking functionality required by the users. UP represents an iterative and cyclic process of software development consisting of specifying use cases, (re)designing the software architecture, and implementing the use cases. UP constitutes a formal and reproducible approach for developing software.

8.1.2 The required functionality

In section 5.3 of this thesis the *functionality listing* of two use cases are analyzed. Since the type of functionality required in the various use cases (listed in Table 5-3) is to a considerable extent the same in all use cases, the functionality listing (cf. Table 5-9) of the use cases 'LE2' and 'LE3 ext' can be taken as representative for all use cases. By specifying which GIS functionality in the analyzed use cases is required from the GIS subsystem or more specifically from the GIS legacy system, this subsection addresses research question 2.

At least until this phase of the IPODLAS system development, all domain-typical functionality can be provided by the legacy systems of the respective subsystem. This means that the the requirements of the analyzed use cases for temporal simulation, visualization, and spatial functionality can be provided by standard TSS, VR, and GIS legacy systems. When addressing research question 2, the spatial functionality required from the IPODLAS system can be classified using the taxonomy of Albrecht (1997); the only exception to this is the spatial simulation functionality (i.e. wind and fire spread simulation), which is also required from the GIS legacy system.

The main type of functionality that is required by the IPODLAS system and that cannot be provided by the legacy systems is *communication functionality*. To provide the domain-typical functionality of the legacy systems to the IPODLAS GUI and thus to the user, the IPODLAS system must be able to remotely control the legacy systems and to access their requested functionality. It must coordinate the application of its embedded legacy systems, access their requested functionality and transfer the output to the appropriate place in the system.

Since most TSS, VR, and GIS legacy systems have been designed as *standalone application*s, remote control and functionality access from beyond system borders is not implemented as standard functionality. Many studies claim that traditional GIS tend to be closed and monolithic (Bergmann et al., 2000b; Bernard, 2001; Preston et al., 2003; Wytzisk, 2003) which makes it hard to provide the existing GIS functionality over system borders. Only in recent times, there have been approaches to provide GIS functionality via a GIS server, e.g. the ArcGIS Server (Frehner et al., 2004; Shi, 2004) or the GRASS server (Blazek and Nardelli, 2004), or over the internet (Babu, 2003; Chang and Park, 2006; Kim and Kim, 2002). Analyzing existing approaches of interoperable and distributed GIS, Bergmann et al. (2000b) states that the ability of interaction of the components through information exchange, in particular seamless data access and access to remote methods is a major requirement of moving towards interoperable GIS.

Since remote control and access of functionality from beyond system borders is not supported by most current TSS, VR, and GIS applications, the main challenge for the IPODLAS system is therefore at this stage of development to bridge this gap and to enable a user to apply the appropriate functionality of the involved legacy systems to fulfill her/his task(s).

8.2 Bringing TSS, VR, and GIS together

This section discusses research subprojects and preliminary IPODLAS prototypes conducted within the IPODLAS project, which are described in chapter 6. On the one hand, the research subprojects and prototypes illustrate the development methodology addressed in resarch question 1 of this thesis by using the iterative and incremental approach of the UP applied in the IPODLAS approach and realizing use cases specified within the IPODLAS framework. On the other hand, the subprojects and prototypes present insights and findings applied for the development of the final IPODLAS system,

which addresses research question 3. These insights and findings are evaluated in section 8.3.

The idea of compensating lacking functionality in one application by using the required functionality of another/other application(s) is straightforward and discussed broadly in the literature for the combination of TSS and GIS (Bernard, 2001; Brimicombe, 2003; Fedra, 1993; Fedra, 1996; Goodchild, 1996; Nyerges, 1993; Raper and Livingstone, 1995) and for the combination of VR and GIS (Gold et al., 2004; Huang et al., 2001; Lindstrom et al., 1998; Pajarola and Widmayer, 2001). In planning and scenario generation, the combination of TSS and VR is used (Wang, 2004). Much less common is the combined usage of applications of all three domains (Wang, 2004); examples include Camara et al. (1998) and Wang (2004). When coupling two or three applications from the domains TSS, VR, and GIS, in general the GIS application provides a platform for data integration, model parameter determination and cartographic visualization. TSS provides temporal capabilities and allows the GIS to go beyond inventory and thematic mapping (Sui and Maggio, 1999). VR offers photorealistic, dynamic, and interactive 3-D visualization and allows interactive exploration of temporal and spatial relationships (Camara et al., 1998). The application of functionality of all three domains and particularly the integration of the functionality in one tool may significantly facilitate the communication of simulation results in a geographic context to the general public (Wang, 2004).

8.2.1 The added value of the combined usage of TSS, VR, and GIS

The two research projects conducted within the IPODLAS project and addressed in this subsection discuss the added value of the combined application of functionality of TSS and GIS regarding the ability to model aspects of LBM dynamics.

LBM-GIS: an LBM migration model

In chapter 4 of Price (2005) the influence of spatial data on the results of TSS simulation is investigated. The implementation of the LBM dynamics model LBM-M9 (Fischlin, 1982) in the TSS legacy system RAMSES (Fischlin, 1982) applies spatial data only in a coarse, parameterized manner. LBM-M9 implemented in RAMSES is compared with the LBM dynamics implemented in LBM-GIS (Price, 2005), which uses fine-graided topographical data to enhance LBM-M9 migration results with spatial functionality (Price, 2005).

The LBM-GIS does not model actual LBM densities significantly better than LBM-M9, neither across space nor in time. However, the consideration of topography in the LBM-GIS allows more detailed predictions of the spatial aspect of spatiotemporal LBM dynamics (Price, 2005). LBM-GIS models potential flight areas for particular sites and environmental conditions and thus enables the determination of potential migration paths, which is not possible within LBM-M9 alone. Hence, LBM-GIS overcomes limitations of LBM-M9, which applies a fixed coarse spatial resolution. LBM-GIS allows the investigation of the influence of topography on migration pattern on a much higher spatial resolution (Price, 2005). Additionally, using spatial data from different regions and several spatial resolutions the concepts of LBM-GIS can be applied for the investigation of LBM dynamics in arbitrary regions using variable spatial grain.

Using the 'maturing typology' of Brimicombe (2003) (cf. subsection 3.4.2) LBM-GIS can be classified into the 'embedded coupling' type. Rules modeling the migration in LBM-M9 have been reimplemented in LBM-GIS, thus establishing a new TSS model "inside" the GIS GRASS using r.mapcalc[1], the GRASS implementation of Map Algebra

[1] http://grass.itc.it/grass60/manuals/html60_user/r.mapcalc.html (accessed April 17, 2006)

(Tomlin, 1990). This prototype was designed only to be applied for the particular research questions of Price (2005). The application of LBM-GIS for advanced (LBM) research questions is limited, since GRASS like other standard GIS does not support the computation of all temporal relationships required for the complete computation of LBM dynamics modeled in LBM-M9 (Price, 2005). A more powerful solution must fully combine temporal and spatial modeling, for example applying the 'integrated coupling' model of the 'maturing typology'. This requires tightly coupled TSS and GIS legacy systems realizing two-way communication on the level of functions. In this approach both legacy systems can access the data and the individual functionalities of the other legacy system. This means, for instance, that the TSS can apply spatial analysis functionality of the GIS — in this case the calculation of elevation, slope, and wind — *directly during* the simulation of LBM dynamics. Currently, this approach is hampered by the stand-alone characteristics of the legacy systems and the isolated nature of the LBM simulation model.

The cross-scale approach

Price et al. (submitted) study the influence of spatial scale on the results of ecological models. They compare simulation results of LBM-M9 (Fischlin, 1982) which considers the *site* as smallest spatial grain with LBM-M11 (Price, 2005) applying the *forest compartment* as smallest spatial grain (cf. subsection 5.1.1). They suggest that there exists an optimal spatial scale respectively spatial grain for modeling LBM dynamics within the Upper Engadine: a further increase of the spatial grain of the input data may not produce a significantly better ability to model LBM dynamics (Price, 2005).

LBM-M11 requires two types of spatial input data: wind statistics and neighborhood statistics (cf. Table 6-1). The data exchange between LBM-M11 and the GRASS GIS is realized as 'one-way data transfer' according to the 'maturing typology'. The GIS is applied to calculate wind and neighborhood statistics in a format that LBM-M11 can read. Since these data exchanges were necessary only once or a few times, they were not implemented as automated data exchange. The neighborhood statistics can be assumed to be static over several years and must therefore be computed only once, while the yearly wind statistics, which represent the yearly wind frequencies, were only required (and exchanged) for a few years. This solution of the 'one-way data transfer' type was chosen, since LBM-M11 was only applied for the spatial scale of forest compartments in the Upper Engadine and for the LBM simulation of a few years. If LBM-M11 is extended to use an arbitrary spatial resolution or if wind statistics for arbitrary years are to be generated "on-the-fly", automated data exchange on the 'loose coupling' level of the 'maturing typology' may be beneficial.

The neighborhood statistics required by LBM-M11 was generated through determination of the nearest center of a polygon in a given direction from any polygon. The neighborhood statistics can be calculated by generating rasters consisting of only the points of interest and then using the respective Map Algebra implementation of the GIS applied. A computationally more efficient way is to extract the coordinates from the GIS-DB in the cartesian Swiss reference system CH1903 and apply (outside of the GIS) a standard programing language to carry out the required calculations.

8.2.2 Iterative development of the software architecture

The prototypes described in this subsection illustrate the development of important aspects of the software architecture of the IPODLAS system. Relevant characteristics of the prototypes are analyzed according to the classification schemes described in section 3.4.

GML 3 for describing spatiotemporal data

The focus of this subproject was on investigating the capabilities of GML 3 for spatiotemporal data representation. The encoding of data by the GIS legacy system through GML is independent of the platform, operating system, language, and the data transfer protocol. The use of a standard data exchange format such as GML enhances the interoperability of the data[2] (Lake et al., 2004) compared to the use of a conversion between various proprietary data formats. Parsers can validate the data structure of the GML document applying the associated XML Schema. The XML characteristics of GML enhance the expressiveness of the data and hence facilitate information exchange (Lake et al., 2004) between *a priori* non-compatible systems such as GIS and TSS. As a member of the XML familily GML exhibits an open structure providing and easing the possibility of further enhancement (Hoheisel, 2002; Jones and Drake, 2002) of the GML structure and thus, of the IPODLAS system. However, an increased data volume due to the tag structure has to be accepted; this drawback can be minimized by compressing the exchanged data (Hoheisel, 2002).

Compared to previous versions, GML 3 provides temporal extensions covering events, histories, and timestamps (Lake, 2001; Lake et al., 2004). The prototype described in subsection 6.2.1 exploits the temporal capabilities of GML 3 to represent the phenomenon of interest — the representation of sites in the Upper Engadine through time — in a more object-based manner. While in GML 2 non-temporal data is replicated in a series of GML documents, the representation in GML 3 using a unified representation provides benefits in data consistency and disk usage compared to the usage of GML 2 documents.

Since the VR legacy system does not support GML and thus the GML data cannot be visualized by the VR legacy system, the generated GML 3 documents are converted to SVG. This format is favourable for visualizing GML vector data, since no information is lost when encoding XML data in SVG, which is designed to represent vector data. Furthermore, XSL provide standard mechanisms to convert GML in SVG (Kiwon et al., 2003). SVG is particularly suited to visualize GML 3 data due to the support of SVG to display the temporal attributes of GML 3.

The ‘intelligent tree’

This prototype was developed to gain initial experience in coordinating and synchronizing the three subsystems TSS, VR, and GIS. The ‘intelligent tree’ applies no central synchronization mechanism unlike the final prototype described in chapter 7. The synchronization functionality, which is for the ‘intelligent tree’ conceptually the ability to generate and delete the semaphore file, is deployed in a decentralized manner on the subsystems. This prototype exhibits aspects of the ‘loose coupling’ type of the ‘maturing

[2] At least if only the subset of the OGC Simple Feature Specification (http://portal.opengeospatial.org/modules/admin/license_agreement.php?suppressHeaders=0&access_license_id=3&target=http://portal.opengeospatial.org/files/index.php?artifact_id=13227, accessed May 2, 2006) is applied. This subset is mostly implemented in a consistent manner in standard GIS and thus can be exchanged.

typology' (cf. subsection 3.4.2): the data models, functionalities, and UIs of the subsystems remain separate. The information exchange between the subsystems of the 'intelligent tree' is realized in a file-based manner and can be seen as the 'linkages between programs' type of the classification of Westervelt and Shapiro (2000), where the output of one system is the input for the next. The 'intelligent tree' illustrated that the synchronization mechanism using semaphore files is a straightforward and simple solution, but becomes quickly limiting with increasing complexity of the process to be synchronized. Moreover, without applying additional functionality for transfering files between the subsystems, this approach is limited to a system where all subsystems have access to a common storage.

Cross-implementation

The 'cross-implementation' prototype was implemented to demonstrate the strenghts and limitations of the respective subsystems. The main outcome was that both legacy systems (TSS and GIS) deal well with the problems they are designed for while problems occur when trying to conduct research not explicitly supported by the legacy systems (cf. 5.1.2.). For instance, on the one hand, simple spatial data structures such as a raster or spatial functionality such as Map Algebra (Tomlin, 1990) required for the implementation of a Wildland fire spread are missing in the applied TSS legacy system (Fischlin, 1991). On the other hand, the applied GIS legacy system does not support temporal functionality (Isenegger et al., 2004).

These findings make a case for the hypothesis that when doing joint research each legacy system can bring in its strengths and avoid its weaknesses. Therefore, the IPODLAS system can benefit from the complementary capabilities of the respective legacy systems (Isenegger et al., 2004). The two prototypes developed in the 'cross-implementation' project apply the 'embedded coupling' type of the 'maturing typology'. In both prototypes, the simulation models are implemented using a modeling language supported by the respective legacy system.

Remote Ramses

'Remote Ramses' (Bergamin, 2004) adds network communication functionality to RAMSES (Fischlin, 1991) — the TSS legacy system of the IPODLAS system — so that RAMSES can be applied as remote simulation server. In contrast to the batch-oriented RASS (Thöny et al., 1995) 'Remote Ramses' supports the interactive capabilities of RAMSES (Bergamin, 2004). 'Remote Ramses' constitutes an important part in the IPODLAS system by providing the only remote, platform-independent communication mechanism with RAMSES. 'Remote Ramses' acts therefore as interface between RAMSES and the rest of the IPODLAS system.

The combination approach of 'Remote Ramses' corresponds with the 'loose coupling' type of the 'maturing typology' of Brimicombe (2003) establishing an automated, two-way data exchange, but keeping 'Remote Ramses' and RAMSES components apart. The control flow and the UI are designed using the 'client-server' solution described in Wittmann (Wittmann, 2000) (cf. subsection 3.4.1), where the user controls the whole system over the UI of only one subsystem. The data exchange applies both, file-based and process-based mechanisms.

The GUI2VR prototype

The GUI2VR prototype has been developed to establish a communication model between the GUI and the rest of the IPODLAS system. The prototype applies a simplified version of the communication mechanisms of the IPODLAS system described in chapter 7. Applying sockets platform-independent communication mechanisms between the subsystems are used, while internally to the subsystems various, partly platform-dependent communication mechanisms are used.

Similar to 'Remote Ramses' the GUI2VR prototype applies the 'loose coupling' type of the 'maturing typology' with automated file-based and process-based, two-way data exchange. In contrast to 'Remote Ramses', both subsystems are visible to the user. While GUI2VR is controlled normally only by the IPODLAS GUI, for the communication with the user both subsystems are applied. The user configures the animation in the GUI subsystem, whereas the spatial navigation and the actual visualization is realized in the VR subsystem, i.e. using the VR legacy system GUI.

8.3 The IPODLAS system

This section discusses the considerations about and the specifications of the IPODLAS system and its development described in chapter 7. Evaluating issues of the software architecture research question 3 of this thesis is addressed. In the following, the term 'IPODLAS system' is used to denote the final IPODLAS prototype detailed in chapter 7.

8.3.1 User interface design

The user interface design applies insights gained from the use case model (cf. subsection 5.2.1). On the level of 'system services' of the 'user interface design level' classification of Löwgren (1993), two main types of prospective usages and thus views of the IPODLAS system can be identified. The pilot users perceive the IPODLAS system as an information system which helps them to acquire and browse through information. The expert users additionally also recognize the research capabilities, for instance for information generation and evaluation. The user interface (UI) of the IPODLAS system is developed considering (and thus reflecting) the main types of users and usages of the IPODLAS system. The specification of the UI on the level of the systems services illustrates the bipartite nature of the IPODLAS GUI (cf. subsection 7.1.1): browsing and collecting information vs. generating and analysing information.

Applying these specifications on the next lower level 'User's model and metaphor' of the 'user interface design level' classification the appropriate user's model(s) and metaphor(s) can be developed (cf. subsection 7.1.2). The user's model and the metaphor applied help the designer of the system to specify the users' view(s) of the system and thus support the design of the system so that it complies with these views. The metaphor used in the development of the IPODLAS system was the one of a flight simulator. This metaphor transports the idea of a IPODLAS system which can be used by pilot users for flying through a (virtual) landscape, increasing the flying altitude to gain a better overview and decreasing the flying altitude to investigate a specific phenomenon of interest from up close. Expert users additionally apply additional functionality in the flight simulator to conduct advanced research such as scenario building.

In particular in the case of a user-driven system such as the IPODLAS system, where all system activity is triggered by the user, the design of the GUI specifying user interaction possibilities at a specific state of a use case is beneficial for the development of the system. The collection and specification of possible user interactions for all states in the state diagram defines the range of events which must be dealt with by the IPODLAS system at the respective states. The state diagram shows that in some states

only file dialog operations such as 'save as...' must be handled by the IPODLAS system, while in other states additionally spatial navigation, view configuration, and animation or simulation operations must be considered.

8.3.2 Software architecture

This subsection addresses the sections 7.2 and 7.3 together discussing concepts detailed in section 7.2 and using examples described in section 7.3.

Nonfunctional requirements

The nonfunctional requirements of the IPODLAS system described in subsection 7.2.1 are discussed in this paragraph. The IPODLAS system is *user-driven*, since the IPODLAS system is triggered only by actions of the user. TwistedMatrix (Fettig, 2005) provides the IPODLAS system with asynchronous communication functionality, i.e to react to a request in a immediate, non-blocking manner. Thus, the user can communicate *interactively* with the IPODLAS system meaning the user is in full control of the IPODLAS system at all times. *Real-time behavior* is in particular mandatory for the VR subsystem, that is, the VR legacy system must react immediately to user interactions. At this phase of development of the IPODLAS system, the user interaction with the VR subsystem is limited to spatial navigation and visualization. This is conducted directly via the VR legacy system user interface, which supports real-time interaction. For instance, the wildland fire spread in the use case 'LE3 ext' (cf. subsection 7.3) triggered by the user in the IPODLAS GUI and calculated in the GIS legacy system can be investigated on the GUI of the VR legacy system in real-time by zooming in and out, choosing different viewing angles, etc.

The software architecture of the IPODLAS system supports *maintenance*; du to the modular structure changes are contained to the affected subsystem(s). The IPODLAS system can be *extended* straightforwardly by implementing more functionality from the integrated legacy systems. An extension of functionality by adding new legacy systems is allowed by the software architecture in the sense that the number of involved subsystems is not limited by architectural constraints. Furthermore, the main communication functionality does not have to be changed when adding new subsystems; the new subsystems only have to be registered in the IPODLAS kernel. The same considerations are also applicable for the *scalability* of the IPODLAS system.

The communication and synchronization concepts do not rely on proprietary features of an individual legacy system. Since only *open source software* is used in the IPODLAS system, all source code is available and can be modified.

Combination typologies

By integrating the three legacy systems TSS, GIS, and VR into the IPODLAS system and adding the subsystems IPODLAS GUI and IPODLAS storage, the IPODLAS system shows aspects of the level of 'tool coupling' according to the 'maturing typology' of Brimicombe (2003). The IPODLAS system is a networked framework having integral subsystems each providing specific services to service requesters. In the IPODLAS system, this means that the TSS subsystem offers its temporal simulation capabilities, the VR subsystem its visualization and user interaction skills, while the GIS subsystem provides spatial functionality. All subsystems within the framework are wrapped by a common user interface and can access the common storage and thus share data. Through the common user interface all subsystems can be controlled and accessed by the user.

Applying the classification of Rhyne (1997) (cf. subsection 3.4.1), the IPODLAS system reaches the 'functional combination level' seeing the IPODLAS GUI as

subsystem controlling all other subsytems via the kernel. The IPODLAS kernel provides the IPODLAS GUI transparent access to all functionality of the subsystems. The IPODLAS GUI is not concerned with the provider of the functionality nor its location. Furthermore, the common IPODLAS storage provides data access for all subsystems of the IPODLAS system.

The IPODLAS system exhibits aspects of two classes of the typology of Westervelt and Shapiro (2000) (cf. subsection 3.4.2). On the one hand, the IPODLAS system shows attributes of the 'shared assets and procedures' class. Regarding the IPODLAS system as the common environment of the subsystems the subsystems get their inputs from a common data and software environment, which supports the establishment of a common UI (the IPODLAS GUI respectively the common communication protocol between the subsystems and the kernel) and a common data storage. On the other hand, the IPODLAS system presents aspects of the 'linkages with distributed objects' class. Focusing on the communication and synchronization aspect, the IPODLAS system provides a common execution environment of the subsystems allowing the kernel to control and thus synchronize the subsystems in a distributed software environment.

Architecture and IPODLAS kernel

In the IPODLAS system all subsystem may need to communicate with each other subsystem. Thus, the *blackboard architecture* has been applied by designating the IPODLAS kernel as communication interface over which all subsystems communicate.

Focusing on the kernel, the IPODLAS system can be seen as a *three-tiered architecture* (cf. subsection 2.2.1). The IPODLAS GUI is then perceived as the client tier, where the user can initiate a service request. Next, the request is received by the second tier, which is in this case formed by the IPODLAS kernel and the communication functionality used for exchanging the messages. The kernel is a mediating kernel which analyzes and breaks down the request in subrequests if necessary, and dispatches the (sub)request to the to the appropriate subsystem(s) — the TSS, VR, or GIS subsystem — which form the application tier. The set of the communication messages constitute a minimal data model which all subsystems share generating interoperability on the level of data exchange formats (cf. section 2.3).

The addition of an intermediate layer is fundamental to three-tiered architectures such as CORBA (OMG, 1999). The coupling of different GIS or DBMS (Bergmann et al., 2000a; Bergmann et al., 2000b; Preston et al., 2003) are examples of three-tiered architectures within the GIS domain. The second tier provides a communication model that is the basis for interactions between the client tier and the application tier. Additionally, the second tier decouples the client tier from heterogeneous legacy systems located in the application tier (Gronmo et al., 2000). In contrast to other three-tiered architectures the current IPODLAS system does not support multi-user mode, i.e. serving multiple clients.

Mediator-based systems are three-tiered systems which can deal with an arbitrary number of relatively autonomous subsystems communicating with each other over a common protocol. A mediator may have multiple standards to access subsystems on the third tier, but presents a single interface to the client situated on the first tier. The second tier of mediator-based, three-tiered systems provides the processing and dispatching of requests from the first tier to the third tier and back (Wiederhold, 1992; Wiederhold, 1995). Being three-tiered architectures, mediating systems exhibit the same advantages. These systems show high scalability and modularity, i.e. a limited amount of clearly defined interfaces between the involved subsystems. Changes in the communication mode between the subsystems or the modification of a subsystem are limited to affected

subsystem(s) and the mediating subsystem (Zaslavsky et al., 2000). The synchronization subsystem within the IPODLAS system — the IPODLAS kernel — limits as the only communication interface between the subsystems the number of interfaces within the IPODLAS system. Therefore, changes in the interaction of the subsystems or even the exchange of a subsystem, for example the use of another GIS, need only to be registered in the IPODLAS kernel (Isenegger et al., 2005).

In the spatial domain mediator-based systems are often applied when integrating different data sources into a GIS. The mediator is used to analyze and break down queries or requests into subqueries and to dispatch these (sub)queries to the appropriate heterogeneous spatial data sources (Savary and Zeitouni, 2003; Zaslavsky et al., 2000). The IPODLAS kernel receives and analyzes requests coming from the IPODLAS GUI respectively from the user. If necessary, a request is broken down in subrequests which can be handled by one subsystem. Table 7-3 shows the requests and subrequests of the use case 'LE3 ext', for instance, the request `simulLBM` which causes the IPODLAS kernel to generate the subrequests `displayInGUI` and `visuLBM` (cf. Figure 7-10, Figure 7-11, and Figure 7-12).

The 'Computer Assisted Protective Action Recommendations System' (CAPARS) of Hodgin et al. (1997) (Brandmeyer and Karimi, 2000) the IPODLAS system constitutes of a framework which complies to the 'tool coupling' level of the 'maturing typology' and which consists of a subsystem coordinating the other subsystems within the framework.

Common UI

Working with the common user interface of the IPODLAS system, the user is decoupled from direct interaction with the legacy subsystems. Thus, the user does not have to know the exact sequence of interactions and conversions required between the legacy systems (Wittmann, 2000), but can concentrate on her/his specific task. The IPODLAS system hides the workflow, information exchange, and required conversion and mappings to a large degree; the users just apply the most appropriate data and functionality the IPODLAS system offers. The common user interface requires seamless access to the functionality of all subsystems, which is provided by the IPODLAS system.

Communication

The communication in the IPODLAS system is realized in a file-based and process-based mode. While within the subsystems the information exchange is partially implemented by file exchange, the communication between the subsystems is established by socket connections based on message passing.

The IPODLAS system applies functionality of the TwistedMatrix framework (Fettig, 2005) exploiting TwistedMatrix support for nonblocking asynchronous servers. Nonblocking asynchronous communication is suited for servers processing many connections requiring little server-side processing, e.g. web or FTP server (Goerzen, 2004). In the implementation of the use case 'LE3 ext' in section 7.3 it is shown that also the IPODLAS system entails to a considerable extent a lot of network transfer and only a limited number of time-consuming legacy system processing tasks.

The functionality to handle the technical aspect of the socket communication of the IPODLAS system is provided by classes of TwistedMatrix: for instance, in case of a failure of a socket connection, TwistedMatrix automatically reestablishes the failed socket connection. By deriving from classes of TwistedMatrix the IPODLAS system applies TwistedMatrix functionality to provide the required socket connections between members of the IPODLAS system, i.e. between the `IpodlasKernel` on the kernel and the `IpodlasClient` on the individual subsystems.

This purely socket-based solution is preferred to a http-based approach, since the IPODLAS project team wanted to avoid the relative inefficiency of the http protocol when dealing with data transmission. This limitation of http-based information transmission was originally considered important in the design phase of the IPODLAS system, in anticipation of the large amount of data (often binary encoded) that are typically found when integrating GIS. However, for the final IPODLAS system, which only exchanges XML-encoded data, this limitation is no longer of crucial importance. Furthermore, a http-based solution would enhance the interoperability by applying widely accepted standards.

On the conceptual level, the IPODLAS system applies the message passing concept due to its support of asynchronous communication (in contrast to e.g. RPC, cf. subsection 2.4.1). The messages used in the IPODLAS system — `ipodlasMessage` (`IM`) and `iData` — are encoded in XML for various reasons: XML is designed for the implementation-neutral exchange of data (Gronmo et al., 2000); XML is platform, software, and hardware-independent; and enhances the expressiveness of the data (Harold and Means, 2004). Moreover, XML is extensible in terms of adapting existing `IMs`, which is beneficial for further developments of the IPODLAS system. Furtermore, XML supports validating of the XML document using the associated XML Schema. The `IMs` establish a protocol for the communication within the IPODLAS system, which each subsystem must support. Thus, the `IMs` respectively the protocol define the range of functionality provided by the IPODLAS system and the information required to be exchanged by the subsystems. Table 7-3 and Table 7-4 (cf. subsection 7.2.3) lists the `IMs` respectively the `iDatas` required to implement the use case 'LE3 ext'.

Deployment, modularity, and storage

For each subsystem of the IPODLAS system there is an `IpodlasClient` instance which communicates with its associated `IpodlasKernel` instance deployed on the synchronizing subsystem, the IPODLAS kernel. The deployment of the instances support the scalability and extensibility of the IPODLAS system: When adding a new subsystem to the IPODLAS system, only an `IpodlasClient` instance on the subsystem needs to be added and an associated `IpodlasKernel` instance on the IPODLAS kernel has to be instantiated. The communication model of the IPODLAS system must not be changed.

The connection between the `IpodlasKernel` and the `IpodlasClients` forms the only interface between the subsystems of the IPODLAS system establishing thus a highly decoupled and modular structure. Due to the modular architecture of the system a stepwise refinement and enhancement of the system can be achieved, which allows for the separate development of different aspects and therefore a smooth interaction of subsystems that are in different phases of their development. Another benefit of a modular design is the enhanced reusability (Preston et al., 2003), extensibility, and scalability of the system (Bergmann et al., 2000b; Wang, 2000). Similar to Bergmann (2000a) and Bernard and Krueger (2000) the layered architecture of the IPODLAS system limits the interdependencies between the subsystems.

The IPODLAS system applies a mixed storage strategy. Slightly simplifying the concept described in subsection 7.2.2 data required by only in one subsystem is stored locally to the respective subsystem in the appropriate format(s), while data potentially used in more than one subsystem is hold in a common format in the central IPODLAS storage. This is the case, for instance, for data produced by the TSS legacy system: instead of executing immediately each potentially time-consuming simulation run requested by the user, the IPODLAS system first checks if the requested simulation data already exists in the IPODLAS storage. The locally stored data in various formats

reduces the required data transfer and enhance the data access. The commonly used data on the IPODLAS storage provides data consistency within the IPODLAS system.

Interoperability

The IPODLAS system supports interoperability by using technological standard means, e.g applying the socket interface to connect the individual machines or XML to exchange information. Moreover, the concepts developed are independent of any specific language, DCP, or legacy system. This increases the general applicability of the concepts and enhances the reusability and extensibility of the developed prototype.

Comparison with other software architectures

Moving the IPODLAS system from interoperability standards situated on the technical level (e.g. the socket interface) and applying standard middleware such as CORBA as communication framework as for instance in the DISGIS project (Gronmo et al., 2000) or web services as in Chang (2006) or Riedemann (2003) would enhance the interoperability and extensibility of the IPODLAS system significantly. Beeing aware of differences in dimensions, maturity, acceptance, and scope of CORBA and web services compared to the IPODLAS system, some properties may be compared nevertheless.

The core functionality that the ORB offers to CORBA (Wytzisk, 2003) is offered by the IPODLAS kernel to the IPODLAS system. The IPODLAS kernel receives and dispatches service requests from the clients to the appropriate legacy system providing the client access and location transparency. While CORBA does this by providing the APIs of potentially remote objects (Wytzisk, 2003) the IPODLAS system uses the protocol established by `IMs`.

The motivation of web services — to integrate existing legacy systems under consideration of their heterogeneous nature in particular in terms of the communication model (Curbera et al., 2001) — is similar to the motivation of the IPODLAS project. To deal with interoperability the web service model is built around XML and web services use only interoperable WWW standards as HTTP or SOAP (Wytzisk, 2003). In the IPODLAS system information between the subsystems is exchanged by XML-encoded `IMs`. The `IMs` are applied in a similar manner as SOAP, where XML messages are contained in an envelope exhibiting a header and a body (Savary and Zeitouni, 2003). Moreover, some functionalities of the IPODLAS system such as a TSS simulation result can easily be provided XML-encoded. On the technical level the IPODLAS system uses (instead of http) "only" socket connections established by TwistedMatrix. Similar as in web services (Curbera et al., 2001) the IPODLAS system uses messages instead of APIs to exchange information and the focus is on delivering services instead of delivering data. A main constituent of web services is not considered in the IPODLAS system. It does not support unified representation of applications and a decentralized usage model, which is provided by IDL respectively by the functional description and localicazation of web services using WSDL, UDDI and other specifications. While web service use a "loosely coupled interaction model" (Curbera et al., 2001, p. 3) that can be considered to comply to the 'loose coupling' level of the 'maturing typology', the IPODLAS system shows aspects of the 'tool coupling' by providing various networked services. Interoperable GIS applications use on the technical level distributed component models such as CORBA (Babu, 2003; Bergmann et al., 2000a; Bergmann et al., 2000b; Preston et al., 2003) or the web service approach (Babu, 2003; Blazek et al., 2002; Huang et al., 2001) to provide remote access to spatial data and functionality. Thus, when comparing interoperable GIS with the IPODLAS system the same considerations apply for them as for CORBA respectively as for web services.

The IPODLAS system provides similar services for its integrated legacy systems as the *High Level Architecture* (HLA), which is a general architecture for distributed computer simulation systems (cf. subsection 3.1.2). While the HLA runtime infrastructure (RTI) provides interoperability on the the API-level, the IPODLAS system offers interoperable data exchange trough the set of the ipodlasMessages (IM). EODISP[3], which is a partial implementation of HLA, applies wrappers to integrate heterogeneous, non-compatible simulation applications. The `IpodlasClients` provide similar functionality by communicating on the one hand on the local subsystem with the legacy system and on the other hand providing the IPODLAS communication interface for the communication between the IPODLAS subsystems. Analogous to EODISP the IPODLAS system implements data-driven communication where each subsystem is characterized by the data it is producing and by the data it is consuming; the control flow is determined by the request and the arrival of data. In contrast to the IPODLAS system, HLA can manage dynamic collections of simulation applications by supporting the publish-subscribe mechanism, where the simulation applications publish their services and other simulation applications subscribe to this services.

The dominant interoperability approaches of the domains TSS and GIS — HLA and ISO/TC 211 and OGC (cf. subsections 3.1.2 and 3.3.2) respectively — remain limited to their respective domain (Bernard, 2001; Wytzisk, 2003). Only in recent times, promising work dedicated to bridge this gap has been conducted (Bernard, 2001; Schulze et al., 2002; Simonis and Wytzisk, 2003; Wytzisk, 2003).

8.3.3 Modification of the IPODLAS system

The software architecture and the implementation of the IPODLAS system exhibit several disadvantages and properties which are probably not beneficial for an extension of the IPODLAS system. The limitations are on the conceptual and on the technical level. The shortcomings and drawbacks of the software architecture and of the implementation of the current IPODLAS system motivate several improvements of both. Due to the logical coherence of the shortcomings and their improvements, they are addressed together in section 9.3 'Limitations and outlook'.

8.3.4 Performance measurements in the use case 'LE3 ext'

This subsection discusses the performance measurements of the use case 'LE3 ext' detailed in section 7.4. The run time analysis has been conducted to gain insights about run time characteristics of the IPODLAS system. These can be used to identify critical parts of the IPODLAS system. The analysis of the run time characteristics of the current IPODLAS system might gives hints about the run time characteristics of an upscaled IPODLAS system, for instance if the exchange of `IpodlasMessages` (`IM`) may form a bottleneck in an upscaled system. Another issue is the distribution of the run time between the individual subsystems.

The measurement of the transfer times of the `IMs` and the `iDatas` suffers from restricted significance, e.g. due to suboptimal synchronization of the system clocks on some platforms or due to the limited number of measurements. Nevertheless, the results can be used to qualitatively estimate the run time characteristics of the implementation of the use case 'LE3 ext'. Since this use case is a typical representative for use cases developed within the IPODLAS framework, its run time behavior can be assumed to be as typical for those of most use cases.

The analysis of the transfer times of the `IMs` and `iDatas` in subsection 7.4.1 shows that the transfer times are not primarily dependent on the size of the `IMs`,

[3] http://pnp-software.com/eodisp/overview.html (accessed September 20, 2006)

respectively `iDatas` (cf. Table 7-6). This means that at least for the number and size of exchanged `IMs` and `iDatas` in 'LE3 ext' and for the hardware used neither the CPUs of the involved platforms nor the network forms a performance bottleneck. Consequently, it can be speculated that when increasing the amount of `IM` and `iData` exchange and/or the size of `IMs` and `iData` by a limited amount (e.g. doubling the number and the average size of exchanged `IMs` and `iDatas`), it is unlikely that the run time charactistics of an upscaled system change significantly.

For extending the IPODLAS system, it is interesting to know how much of the run time is consumed by the different time consuming subsystems. Subsection 7.4.2 investigates this by distinguishing the time consumers *user*, *legacy system*, and *rest of the IPODLAS system*. It is illustrated in Table 7-7 and Table 7-8 (cf subsection 7.4.2), that the time consumed by the 'rest of the IPODLAS system' in the use case 'LE3 ext' is in the range of one percent of the total run time. Thus, when improving the run time of the IPODLAS system, the largest gain in run time may be achieved by optimizing the run times of the legacy systems.

Summarizing the results of section 7.4, it is suggested that an extension of the IPODLAS system to a limited amount of involved subsystems and functionalities may not be constrained considerabely by the manner of applied information exchange, i.e. using IMs to establish the communication between the subsystems.

9 Conclusions

The dynamic representation of changes in alpine landscapes — the overall goal of IPDOLAS project — is addressed by this thesis through the application of existing knowledge and applications from the domains TSS, VR, and GIS to deal with different aspects of processes forming alpine landscapes. TSS covers the temporal functionality, VR provides user interaction and visualization functionality, and GIS is applied to handle spatial problems. It is suggested that a combined usage of functionality of TSS, VR, and GIS generates different results and may lead to more holistic insights than using functionality from a single domain.

The thesis presents the IPODLAS framework which consists on the one hand of the methodology termed IPODLAS approach and on the other hand the IPODLAS system including concepts and applications to improve the understanding of spatiotemporal and cross-scale environmental processes. By applying the complementary strengths and avoiding the drawbacks of the different domains the functionalities and knowledge from TSS, VR, and GIS can be used for tasks they are suited for. Thus, the IPODLAS system embedds legacy systems from TSS, VR, and GIS and exploits their functionalities. Being part of the IPODLAS project this work in particular covers spatial aspects, thus taking a GIS perspective. In chapter 1 the goals of the thesis were defined as:

1. Specification of a development methodology for the IPODLAS system
2. Identification of the required GIS functionality of the IPODLAS system
3. Specification and design of the software architecture for the IPODLAS system

In the first subsection of this chapter the achievements of the thesis are discussed while in subsection 9.2 the significance of the achievements is critically evaluated. In subsection 9.3 limitations of the results of the thesis are enumerated thus forming the background for the outlook. The thesis closes with the concluding remarks.

9.1 Achievements

The achievements have been accrued with respect to the three objectives listed above. These results are summarized and discussed in the following subsections.

9.1.1 The IPODLAS approach

The first research question of the thesis is addressed by the IPODLAS approach. The IPODLAS approach supports the development of a system — the IPODLAS system — that enhances the handling and representing of spatiotemporal and cross-scale environmental processes. An important achievement of the IPODLAS approach is the condensed and formal collection of user requirments and application knowledge captured in usage scenarios and derivates.

The IPODLAS approach specified a *case study framework* consisting of alpine case studies from different domains providing data and models on different scales. Instead of using only one simulation model for all scales to represent cross-scale processes, several simulation models are applied to represent the respective processes on different scales. Thus, the scale-sensitive modeling of the processes can be supported. Within the IPODLAS framework *use cases* are specified situated in the diverse case studies using concepts of the 'Unified software development process' (UP) (Jacobson et al., 1999). Use cases are an intuitive means for specifying concrete usage scenarios. The *use case model* consisting of all use cases therefore allows to define the complete functionality of the planned system in a clear and concise manner. By specifying each user interaction with the user interface of the IPODLAS system the use cases are refined into the *sequenced*

action lists defining the interaction of the user with the system and the reaction of the system. Graphical definitions of the user interface help to specify the information available to the user and her/his interaction options. The *functionality listings* define for each action of the sequenced action list which functionality is required and which subsystem may provide this functionality: the listings specify the interaction sequence and functional divison of labor on the subsystem level.

Several use cases are realized in subprojects and preliminary prototypes conducted within the IPODLAS project (cf. chapter 6 and section 8.2). According to the cyclic and incremental development mechanisms of the UP findings and insights gained from these subprojects and prototypes helped again in turn to develop the concepts and the IPODLAS system.

9.1.2 The required functionality

The *functionality listings* are the means to specify for each use case which functionality is provided by which subsystem. Thus, they address research question 2 by specifying which funcitonality must be provided by the GIS subsystem. Developed within the IPODLAS approach the functionality listings are derived from the sequenced action lists. Considering the use cases 'LE2' and 'LE3 ext' to be representative for all use cases developed within the IPODLAS project, the functionality listings define which subsystem provides which functionality to satisfy the requirements of the IPODLAS system specified in the use cases 'LE2' and 'LE3 ext', respectively. Analyzing the required GIS functionality these listings can be applied to evaluate whether standard GIS legacy system functionality meets all required GIS functionality.

9.1.3 The IPODLAS system

Research question 3 addresses crucial concepts of the IPODLAS system and its software architecture. The IPODLAS system embedds the TSS, the VR, and the GIS legacy system in potentially distributed subsystems and provides the users smooth and seamless access to their desired functions. Besides wrapping the legacy systems, the core contribution of the IPODLAS system is the asynchrounous communication between distributed processes within a heterogeneous environment and the minimal communication model and protocol.

The IPODLAS system has been implemented as 'proof-of-concept'-prototype applying the appropriate means to efficiently show the viability of the developed concepts and architecture. The use cases 'LE2' and 'LE3 ext' are fully implemented with two exception. Firstly, the the TSS subsystem is not really involved in implementation of the use cases due to limited functionality of the TSS interface 'Remote Ramses' when remotly controling the TSS legacy system RAMSES. As workaround, the IPODLAS system substitutes the interaction with the TSS subsystem by the imitating the data exchange with 'Remote Ramses' applying real data from RAMSES in a static manner (cf. section 7.3). Secondly, the storage strategy described in subsection 7.2.2 is slightly simplified (cf. subsection 8.3.2) by using more than two persistent storages for data only used by one subsystem.

The graphical user interface (GUI)

The IPODLAS GUI allows the user to interact smoothly with all subsystems of the IPODLAS system. The GUI can access functionality of all subsystems, i.e. of all legacy systems, and provides them to the user in one common GUI. Thus, the user is not concerned with conversions between subsystems and their sequence, which would require a sound knowledge of the individual legacy systems. Hence, the user can concentrate on her/his task and is not distracted with technical details. The IPODLAS

GUI was developed to assist in the user interface definition, e.g. to develop the user's model and the state diagram. However, the complete implementation of the IPODLAS GUI was not a primary goal of the thesis.

The IPODLAS GUI has been evolved to support two types of users and thus views of the system. The design of the GUI supports the information acquiring and browsing behavior of the *pilot user* as well as the information generating and evaluating activity of the *expert user*. The metaphor of the flight simulator is applied to convey the image of an easy-to-use GUI which allows intuitive user interaction to investigate the phenomenon of interest.

The *state diagram* was used in the development of the IPODLAS GUI to specify the states of the GUI. The state diagram collects and generalizes the use cases and condenses common sets of user interactions to states of the GUI. Thus, the state diagram specifies the possible interactions of the user with the GUI for each state. Applying these specifications the range of actions is defined which the IPODLAS GUI can trigger and thus the range of actions the IPODLAS system must handle for each state of the GUI.

Software architecture

The software architecture must be designed to support the *functional* and the *nonfunctional requirements*. These requirements have been identified by specifying the use cases within the IPODLAS framework. The functional requirements are captured in the use case model; these requirements specify the functionality the IPODLAS system must provide to satisfy the users's needs defined in the respective use cases. The nonfunctional requirements — user-driven, interactive, and real-time behavior — result from requirements of the users or the designers towards the usage and characteristics of the system being developed. Their consideration results in a system exhibiting asynchronous communication. Additional nonfunctional requirements, i.e. maintainability, extensibility, scalability, and usage of open source software result from demands of the designers towards the IPODLAS system to support its further development.

The IPODLAS system applies the *blackboard architecture* to meet certain communication and synchronization concerns. All subsystems may have to communicate with each other and the interaction of the subsystems follows a certain sequence requiring synchronization. The *IPODLAS kernel* has been designed as "blackboard" forming the only communication interface and centralising the synchronization functionality. The IPODLAS system is scalable and extensible meaning that the number of subsystems communicating over the `IpodlasClient` instances with the associated `IpodlasKernel` instances on the IPODLAS kernel is conceptually not limited.

The IPODLAS system is controlled by the user through the common IPODLAS GUI and all subsystems share data in the IPODLAS storage which can be accessed via the kernel. Thus, the IPODLAS system exhibits aspects of a *three-tiered architecture* with the IPODLAS GUI as client tier, the IPODLAS kernel as second tier, and the legacy systems and the IPODLAS storage as third tier. The kernel and the associated IPODLAS communication functionality provides as second tier the communication model, a minimal common data model and thus interoperability on the data level between the first and the third tier; the kernel presents the IPODLAS GUI a unified interface to the subsystems in the third tier. The kernel is a *mediating kernel*, which receives requests from the first tier, analyzes, breaks them down, and dispatches them to the appropriate subsystem(s) located in the third tier. Having the kernel as the only communication interface the IPODLAS system exhibits a strictly modular and decoupled architecture beneficial for maintaining and extending the system.

The *communication* in the IPODLAS system between the subsystems is established by *asynchronous message passing*. The subsystems send their requests as messages to the kernel; the kernel receives a request, dispatches it and then processes the next request without having to wait for the result of the previous request. The messages exchanged, the `ipodlasMessages (IM)` and the `iDatas`, establish a *communication protocol* which all subsystems must support. On the technical level the communication within the subsystems is realized by sending XML-encoded messages over socket connections established by the *TwistedMatrix framework*. The communication interface is established by `IpodlasClient` and `IpodlasKernel` instances, both are subclasses of TwistedMatrix classes. Each subsystem consists of an `IpodlasClient` instance which exchanges messages via socket connections with the associated `IpodlasKernel` instance deployed on the kernel. The communication within the subsystems is implemented using the appropriate, partly platform-specific mechanisms (cf. chapter 7).

Performance measurements of the use case 'LE3 ext'

The analysis of the run time statistics of the use case 'LE3 ext' shows that the transfer times of the `IMs` and `iDatas` are not primarily correlated with the size of the `IMs` and `iDatas`. Moreover, only about one percent of the total run time of the use case 'LE3 ext' is consumed by the exchanging and managing of `IMs` and `iDatas`, the rest of the time is spent by the legacy systems and on user interactions.

9.2 Insights

9.2.1 The IPODLAS approach

It is suggested that the IPODLAS approach is a methodology that can be applied to develop projects which combine functionalities from different domains to represent the phenomenon of interest on multiple scales. The approach supports a more comprehensive representation of the phenomenon in a spatiotemporal and cross-scale manner compared to using functionality of only one domain and on only one scale. The application of *case studies*, models, and data from domains with complementary foci foster a holistic representation of the observed phenomena.

The application of the UP in the IPODLAS approach supports the implementation of the specified *use cases* situated in the case studies in a software project using a formal and systematic development approach. The iterative and incremental workflow of the UP is a systematic way to move from user requirements to a software system which meets the user requirements (Jacobson et al., 1999). Additionally, the *graphical specification of the GUI* of the use cases and the refinement of the use cases in *sequenced action lists* help to define the user interface and constitute the base for the *functionality listings*. These listings are applied to specify the sequence and the division of labor between the subsystems.

9.2.2 The required functionality

Regarding the functionality required in the specified use cases and thus the functions implemented in the final prototype all *domain-typical functionality* specified in the functionality listings can be provided by the corresponding subsystems respectively legacy systems. For the GIS subsystem this means that the domain-typical functionality required in the use cases, such as spatial and thematic search or locational and terrain analysis, can be provided by standard legacy GIS. Also standard TSS and VR legacy systems can provide the required domain-typical functionality.

To access and use the domain-typical functionality the IPODLAS system apply *communication functionality* (subsection 5.3.2). In contrast to domain-typical functionality,

communication functionality connecting the legacy systems with the IPODLAS system cannot be provided the legacy systems; the functionality required to communicate with the IPODLAS system is not standard functionality of the legacy systems. Thus, the required communcation functionality must be provided by the IPODLAS system; this functionality connecting the legacy systems is an added value of the IPODLAS system compared to the isolated use of the three legacy systems.

This insight is reflected in the sharing of tasks of the IPODLAS system. While the domain-typical functionality is provided by the respective legacy systems, for instance, simulating Larch Bud Moth dynamics or retrieving spatial data, the communication between the subsystems is established by the `IpodlasClient` and `IpodlasKernel` instances deployed on the subsystems.

9.2.3 The IPODLAS system

The graphical user interface (GUI)

The IPODLAS GUI is an important means to decouple the user from the subsystems. Perceiving the IPODLAS system as a system exhibiting a three-tiered architecture, a *common IPODLAS GUI* acts as an application in the client tier hiding and thus decoupling together with the IPODLAS kernel on the second tier the heterogeneity and complexity of the subsystems and their interactions on the third tier.

To present the IPODLAS system as a landscape analysis system the IPODLAS system must be controllable over only one GUI. The user must be able to access and use the functions of the IPODLAS GUI and of the VR legacy system GUI in a common GUI. Only then the intuitive spatial navigation and visualization functionalities of the VR legacy system are joined with the user interaction possibilities of the IPODLAS GUI allowing the investigation of spatiotemporal and cross-scale processes modeled in the TSS and in the GIS subsystem.

Software architecture

The software architecture summarizes the most important structural properties and interfaces of the system by hiding details. The overall goal of the software architecture is to design a system whose major structures are change tolerant and/or change resilient (Jacobson et al., 1999).

The functional requirements describe an action that a software system must be able to perform (Jacobson et al., 1999); they are collected by specifying use cases. The more use cases are specified the more functional requirements are collected. That is, the range of functions that IPODLAS provide to satisfy user requirements correlates roughly the range of implemented use cases. In contrast, the range of nonfunctional requirements specifying the nature and characteristics of the IPODLAS system tend to be more stable. The analysis of the use cases has shown that always the same nonfunctional requirements — user-driven, interactive, and real-time — are requested. It is suggested that when specifying additional use cases only a few new nonfunctional requirements must be added. Hence, the properties and the behavior characteristics of the IPODLAS system (specified by primarily by the nonfunctional requirements) may be not subject to considerable change when implementing more use cases.

The IPODLAS kernel as the "*blackboard*" of the IPODLAS system establishes the communication and synchronization of many subsystems by concentrating the control of the communication in one place. The *mediating* kernel is the place where the request management is accomplished by analyzing, administrating, and dispatching the requests. An event-based architecture would require enhanced synchronization of an arbitrary number of equitable subsystems whose communication model does not follow

a predefined sequence but requires communication mechanisms between each of the participating subsystems.

The *common IPODLAS GUI* frees the user from knowing each subsystem at an advanced level. Instead of having to deal with different user interfaces and data conversions the user can access via the IPODLAS GUI the functions she/he requires to solve her/his needs. In the IPODLAS system the kernel decouples the client tier, the GUI, from the third tier establishing conceptual aspects of a *three-tiered architecture*.

The communication between the subsystems within the IPODLAS system is implemented applying asynchronous message passing. Instead of waiting for the response as in synchronous communication, the kernel dispatches the requests and is ready to process the next request. The TwistedMatrix framework covers the "technical" issues of the asynchronous communication establishing the socket connections between its instances and handling the sending and receiving of data. The IPODLAS system, based upon TwistedMatrix classes, exploits this functionality and can concentrate on the processing of the transmitted data, i.e. the `IMs` and `iDatas`.

The encoding of the `IMs` in XML exhibits two advantages: Firstly, XML is human- and machine-readable faciliating on the one hand the development of the IPODLAS system and on the other hand the parsing of the XML-based messages. Secondly, its structure is extensible meaning that existing `IpodlasMessages` (`IM`) and `iDatas` can be extended easily and new `IMs` and `iDatas` can be added. The implemented use cases define the range of `IM` and `iDatas`, which the IPODLAS subsystems must support; this set of `IMs` and `iDatas` thus constitutes the communication protocol. Altough the current IPODLAS system consists of a relatively small set of messages, some candidates of use case-independent `IMs` and `iDatas` can be identified, for instance 'readControlfile' 'readDataFile', and 'writtenDatafile'.

The software architeture and in particular the communication model was developed to allow combined usage of arbitrary legacy systems of the involved domains. It is therefore not limited to the actual embedded legacy systems, particular platforms or operating systems, or (domain-specific) interoperability initiatives such as HLA and OGC or web services. The non-consideration of these interoperability initiatives originates from historical and conceptual reasons. In the design and development phase of the IPODLAS system these initiatives were not yet in a fully mature state, so no full-fledged applications providing the respective functionality were available. From the conceptual view the IPODLAS system is developed to be a 'proof-of-concept'-prototype. The trade-off of this solution — the relatively efficient realization of the required communication functionality provided by TwistedMatrix and thus avoiding the overhead of having to comply with the major features of the respective standards versus the benefits of functionality provided by applications complying with the major standards — has been considered as appropriate for the development of a prototype.

The IPODLAS system constitutes a simple, operational approach requiring only network connections between the subsystems and an implementation of the open source framework TwistedMatrix, which in turn requires Python to be installed. The `IpodlasClient` and the associated `IpodlasKernel` instances can connect different platforms over the network and communicate within the platform using the appropriate means. Thus, the software architecture entails a communication model suited for distributed systems in a heterogeneous environment. The exclusive use of open source software in the final prototype allows unconstrained use of all legacy systems and other IPODLAS system components and adaptions of their code; hence the IPODLAS system is open and extensible.

Performance measurements in the use case 'LE3 ext'

The performance measurements showed that the exchange of `IMs` and `iDatas` are not a decisive factor for the run time of the use case 'LE3 ext'. Considering the same boundary conditions, it can be suggested that a (limited) extension of the IPODLAS system is not constrained significantly by the communication model established by XML-based message exchanging through sockets.

9.3 Limitations and Outlook

9.3.1 Conceptual challenges

On the conceptual level the IPODLAS framework still faces major challenges to fully reach the goals of the IPODLAS project. The ultimate goal is an IPODLAS system where spatiotemporal and cross-scale environmental processes can be examined by virtually "flying-over" and inspecting them in detail, by zooming in and interactively investigating and controling them. To reach these goals several limitations of the current IPODLAS system must be overcome:

- The subsystems must be able to access functionality of the other subsystems at all time. For instance, in order to truly model a spatiotemporal process the TSS simulation model must be able to access GIS functionality and to use its results at all stages of the simulation model execution. This means that the *integration* must be realized on the *functionality level*, i.e. each subsystem must be able to access and use functionality of others subsystems. Currently, the TSS simulation models only apply spatial data as input data rather than using them during simulation, e.g. in feedback loops. Furthermore, the output of simulations in the IPODLAS system is transferred only at the end of the simulations to other subsystems. To gain maximal flexibility the option to transfer simulation results after each simulation step should be available.
- The *visualization* of cross-scale processes in the VR subsystem should be *scale-sensitive*. That is, when the scale of the visualization changes and reaches a certain threshold data from another, more appropriate simulation model should be applied automatically.
- In order to effectively exchange data a *common understanding* of the exchanged information is required. The subsystems must share a *common data model* on which they map the data model of their embedded legacy system. In the current IPODLAS system, the data models of the legacy systems are only rudimentarily mapped through `IMs` and `iDatas`.
- To enhance the interoperability of the IPODLAS system the *interoperability initiatives* of the involved domains should be considered, i.e. HLA, OGC, and ISO/TC 211. This would advance the IPODLAS system to the interoperability level of APIs and data models. Thus, on the one hand the compliance with interoperability standards would enhance the interoperability of the IPODLAS system and on the other hand the IPODLAS system could use functionality provided by applications complying with the corresponding inititiatives.
- An event-based architecture (cf. subsection 2.2.1) can be an option for future IPODLAS system. Applying this architecture the subsystems react on events from other subsystems transmitted through a bus connecting all subsystems omitting a central synchronizing subsystem. An event in a subsystem is triggered in this case by the action of a user on the GUI or by the message sent from another subsystem.

9.3.2 Technical challenges

When regarding the final IPODLAS system from the technical perspective several limitations can be identified. The functionality provided by the current prototype is limited to the functionality requested in the implemented use cases. To add more functionality to the prototype — conceptually straightforward — more use cases can be implemented providing new functionality of the legacy systems to the IPODLAS system. If the requested functionality cannot be provided by the existing legacy systems or if better implementations of the requested functionality is available in other legacy systems, these legacy systems can replace the existing legacy systems or can be added as additional subsystems. Regarding the software architecture the extension of the IPODLAS system by adding new susbsystems is conceptually not limited.

On the technical level, the IPODLAS system reached a certain development state, where its use as 'proof-of-concept'-prototype has to be compared with the effort necessary to implement more functionality. The architectural structures developed in the IPODLAS system have been considered as appropriate for the current prototype. Fast prototyping was supported due to the relative simplicity of the architecture. However, it is suggested to consider a redesign of the main architectural issues for further development and extension of the IPODLAS system. In the long run, it is recommended to apply standard components to limit the required development efforts and to exploit existing functionality. That is, for instance, using legacy systems and components complying with the main technical and domain-specific interoperability standards reduce the effort required to interoperate with these subsystems. When advancing the current software architecture several drawbacks have to be addressed: the usability of the IPODLAS GUI, the physical separation of the VR GUI and the IPODLAS GUI, the separation of the control and data flow, the communication protocol, and other software architectural issues. Suggested improvements of the current software architecture to overcome the limitations are described in the following.

Replacement of components of the IPODLAS system

The IPODLAS system constitutes a prototype whose software architecture integrates those components which exhibit the requested properties. Due to changing user requirements during the development of a software system and due to new technologies consequentially also the requirements towards the individual components may change. In the following potential candidates for the replacement of the components of the current IPODLAS system are shortly outlined.

Simulink and Scilab (cf. subsection 3.1.4) can be seen as potential alternatives to the TSS legacy system RAMSES. Both provide a broad range of simulation functionality, are platform-independent, and provide (in contrast to RAMSES) interfaces with major programming languages. The VR legacy system VTP could be replaced by the Windows-based LandEx (cf. subsection 3.2.4) supporting a wider range of raster, vector, and 3-D data types. ArcGIS (cf. subsection 3.3.4) is a valid alternative to the current GIS legacy system GRASS provdiding a broad range of GIS software tools. In particular when moving the IPODLAS system towards the web service architecture, the ArcGIS server offering access via a http-interface and the internet map server ArcIMS can contribute valuable functionality. HLA can be applied to replace TwistedMatrix offering advanced distributed system management functionality (cf. subsection 3.1.2 and 8.3.2).

Usability evaluation of the IPODLAS system

The IPODLAS system and in particular the IPODLAS GUI has been developed having the two main types of users — the pilot user and the expert user — in mind. To evaluate

the design and the implementation of the IPODLAS system and of the GUI it must be tested with different types of real users from different research areas.

Questions to be tested are the usability, types of usages, and the range of functionality of the IPODLAS system and GUI. *Usability* refers to if and how the functionality provided by the GUI and thus by the IPODLAS system can be utilized by the users. Test users may perceive the IPODLAS system differently than foreseen by the designers and therefore exhibit different *types of usages* and in consequence require a different GUI and system design. Real test users may claim missing functionality or different characteristics of implemented functionality, thus requesting a different *range of functionality*.

VR GUI and the IPODLAS GUI

While in the preceding paragraph functional aspects of the IPODLAS system and the IPODLAS GUI are addressed, this paragraph focuses on the architectural aspects. In the current IPODLAS system the IPODLAS GUI is deployed physically on another platform than the VR subsystem. Their functionalities are not linked. In more advanced IPODLAS system versions, it may be beneficial for the users if they can interact with only one *common integrated IPODLAS GUI*. This can be achieved by linking the functionality of the current IPODLAS GUI and the VR legacy GUI or by embedding one GUI into the other. The metaphor of the flight simulator applied in the design of the IPODLAS GUI can only be fully exploited when the IPODLAS GUI and the VR legacy system GUI are linked to one common integrated IPODLAS GUI. Then the spatial 3-D navigation of the VR legacy system and the interaction capabilities of the current IPODLAS GUI can be exploited to create the impression of real user interaction with the phenomenon of interest in virtual reality.

Currently, all communication between the subsystems is realized via the IPODLAS kernel (cf. subsection 7.2.2). Due to efficiency considerations the strict modularity of the software architecture of an improved IPODLAS system is broken up to a certain extent. A *direct interface* between the VR subsystem and the IPOPDLAS GUI subystem may be introduced (cf. Figure 9-1). This is done due to potential heavy communication load between the VR subsystem and the IPODLAS GUI subsystem assuming all user interaction and visualization taking place on one common IPODLAS GUI. The real-time requirements of the IPODLAS system apply primarily to the VR subsystem, in particular to the VR legacy system. The user interaction over the common IPODLAS GUI with the VR legacy system (i.e. spatial 3-D navigation) and the movie-like animation of processes require a fast and latency-free connection between the common integrated IPODLAS GUI and the VR legacy system.

Control and data flow

In the current IPODLAS system the control flow and the data flow are handled by the IPODLAS kernel (cf. Figure 7-4). The control and the data flow established by the exchange of `IMs` and `iDatas` respectively is *not separated*: Both use the same communication mechanisms. The `IMs` and the `iDatas` are transmitted by the same socket connections and handled by the same major control structures. With increasing `iData` volume exchanged, the kernel is likely to become a bottleneck if the control and the data flow both are handled via the kernel.

Another weakness of the control and data flow of the current IPODLAS system is the limitation of exchanged data size. The maximum size of a data chunk exchanged in one part over socket connections established by TwistedMatrix is limited to about 16 KB. This limitation is particularly relevant for `iDatas` which are likely to be larger.

In an improved software architecture the kernel establishes only the control flow via the kernel (cf. Figure 9-1); in contrast, the data flow is only managed by the kernel without establishing it. When data is to be transferred between subsystems, the data flow can be organized by the kernel through passing the location of the required data to the requesting subsystem which then itself establishes a direct connection to the target subsystem by-passing the kernel.

Protocol

The collection of `IMs` developed in the current IPODLAS system is at least partly use case-dependent. The implementation of more use cases, in particular use cases situated in other case studies, is likely to generate more and different `IMs`, that is, existing `IMs` and `iDatas` may be redesigned and broken down in smaller, atomic `IMs`. A larger number of `IMs` potentially allow the identification of a number of use case-independent `IMs`. These `IMs` then form the use case-independent part of the communication of the IPODLAS system, thus defining a general use case-independent communication protocol. For parsing the XML-encoded `IMs` and `iDatas`, several parsers provided in most modern programming languages or as stand-alone applications can be used to parse and validate the `IMs` and `iDatas`.

The *feedback* functionality of the current IPODLAS system provides for each `IM` representing a request the generation of an associated `IM` representing a receipt which is sent back to the requestor. This basic functionality can be extended to establish a more advanced feedback functionality. Whenever the user triggers an action on the IPODLAS GUI, the IPODLAS kernel receives this request. The kernel generates a receipt, which is propagated back as primary feedback to the user informing him about the receipt of the request by the IPODLAS system. Meanwhile the kernel analyzes the request, breaks it down into subrequests and sends the subrequests to the appropriate subsystems. The involved subsystems provide an estimate of the time required and begin to process the subrequests. The time estimates are sent back to the kernel. The kernel collects all time estimations, analyzes them and sends the result as secondary feedback to the GUI to inform the user. If this secondary feedback processing is done continuously, it can be displayed in the GUI to the user as progress bar.

In the current IPODLAS system, the `IMs` and the `iDatas` are encoded in a IPODLAS-specific XML format. Using a standard information exchange protocol such as the simple object access protocol (SOAP) (cf. subsection 2.3.2) allows the application of standard communication functionality to process these messages. SOAP could be applied to wrap the content of the `IMs` and to describe the content of the message and its processing (Englander, 2002). When handling `iData`, SOAP supports spatial data encoded in GML (Savary and Zeitouni, 2003).

Storage

The current IPODLAS system stores its data in plain text files in the common IPODLAS storage, thus avoiding data inconsistencies. The IPODLAS storage also provides metadata in plain text describing the data available to the IPODLAS system. For each user request requiring the generation of new data, e.g. the result of a simulation, the metadata of the IPODLAS storage is queried first. If the required data is available, the data is directly delivered from the IPDOLAS storage.

An improved IPODLAS system may employ a *database* (DB) as common storage, which adds functionality to manage interactions between data and processes (Sauer, 1994) such as synchronizing data access. Compared to the current plain text solution the management of the metadata and the linkage to the actual data is supported by the DB functionality.

Due to efficieny considerations it may be beneficial to *closely link* a future IPODLAS storage to the second persistent storage of the IPODLAS system, the GIS storage. If the GIS legacy system supports the storage of data in non-proprietary DBs, the spatial and the non-spatial data can be stored in a unified manner in one DB. This functionality is, for example, provided by ArcGIS using ArcSDE[1] (ESRI, 2005) or GRASS using PostGIS (Blazek et al., 2002).

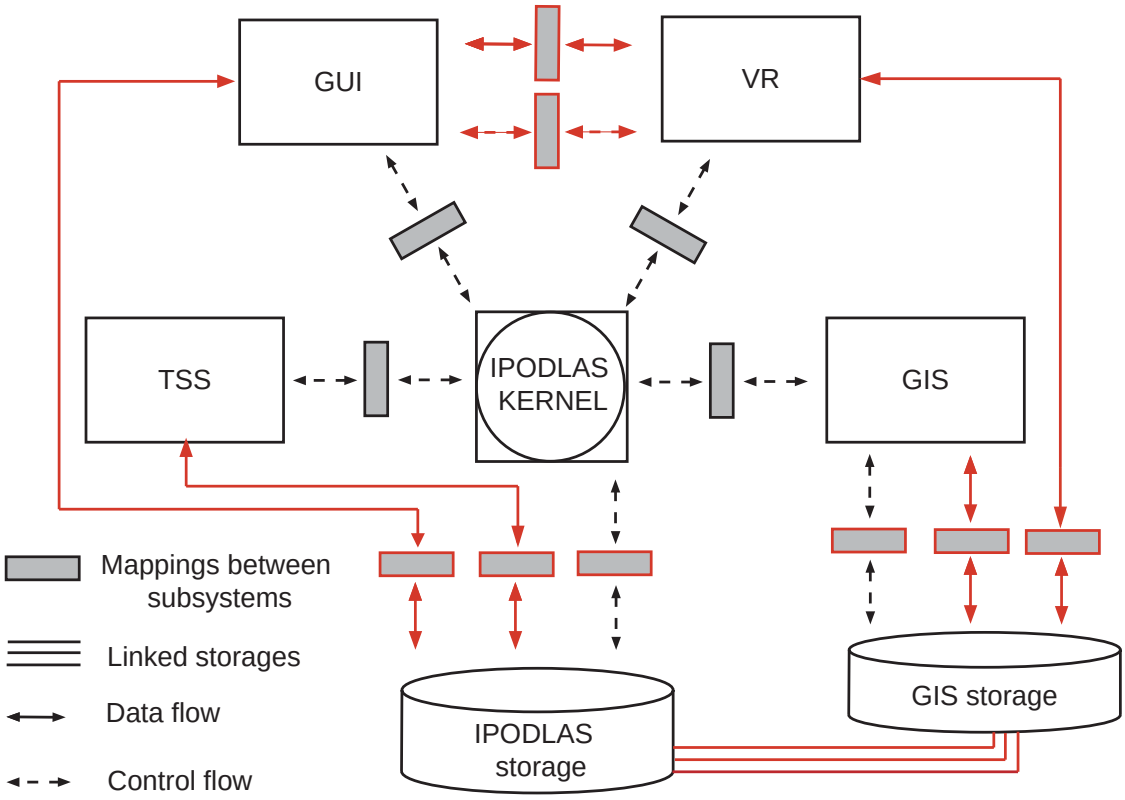

Figure 9-1 A schematic view of the improved IPODLAS software architecture; changes compared to the architecture described in Figure 7-4 are marked in red. The main differences are the direct connection between the GUI and the VR subsystem, the linkage of the two persistent storages, and the separation of the control and data flow. In this improved architecture the kernel establishes only the control flow between the subsystems, while the data flow is realized directly between the subsystems.

Distributed computation and IPODLAS kernel

In the current IPODLAS system, the communication on the technical level between the subsystems, potentially deployed on distributed platforms, is established using socket connections provided by TwistedMatrix (Fettig, 2005). To achieve a higher degree of interoperability and to benefit from their functionality the use of *standard distributed component models* such as CORBA (cf. subsection 2.3.2) with exposed interfaces and hidden implementations (Wytzisk, 2003) may be advisable to connect the subsystems.

The current IPODLAS system stores and thus manages all requests represented by `IMs` in one hash table (cf. subsection 7.2.3). An advanced IPODLAS system may distinguish requests according to their state. For the management of the requests coming from the subsystems the kernel provides three queues: a queue for incoming and not yet processed requests; a queue for requests which are processed momentarily; and a queue for processed requests. Additionally, as the IPOLDAS kernel provides mediating

[1] http://www.esri.com/software/arcgis/arcsde/index.html (accessed May 9, 2006)

functionality, if a request has to be broken down into subrequests, the generated 1:n relation has to be recorded. This relation is useful for the management of the requests and related subrequests. For instance, a request is only completed when all subrequests have been completed. Thus, the IPODLAS system can keep track of the states of all requests.

If standard middleware technology such as CORBA is used to communicate within the future IPODLAS system, the usage of an *application server* may be beneficial (cf. 2.2.1). Most application servers provide functionality that support the concept of middleware-based communication between the kernel and the subsystems. Furthermore, typical application server functionality, which may be useful for the IPODLAS system, are DB access and web server handling (Feiler, 2000).

Web service

To make the IPODLAS functionality available over the internet the current IPODLAS system can be modified to apply the web service architecture (cf. subsection 2.3.2). Web services must be addressable via HTTP over the internet, e.g. via a webserver, and the data exchange is implemented using XML (Curbera et al., 2001). The functionality of an application server integrated in the IPODLAS kernel can be applied to establish the communication between the webserver processing the http connections and the kernel. Moreover, to support automated web service usage the IPODLAS system must represent capabilities and interfaces of offered web services in XML using the web service description language (WSDL) and universal description, discovery, and integration (UDDI) (Riedemann and Timm, 2003). Thus, the IPODLAS system provides a catalog service with metadata describing both, the services offered and the interfaces.

The exchange of data encoded in XML between the IPODLAS system and a client via the internet is straightforward in case of a query for spatial data (e.g. the data can be exchanged GML-encoded), but challenging when services of the VR subsystem are requested. Seeing IPODLAS as a three-tiered system, a fused GUI linking the functionalities of the IPODLAS GUI and the VR legacy GUI could be provided to clients as downloadable application or plug-in. This GUI could be installed on the client's platform establishing as client tier the communication with the IPODLAS kernel applying socket connections.

9.4 Concluding remarks

This thesis and the IPODLAS project in general emerge from the desire to enable comprehensive modeling of spatiotemporal, cross-scale environmental processes. The core idea is the application of functionality of the three domains TSS, VR, and GIS instead of using functionality of only one domain in succession. Exploiting this idea the IPODLAS framework provides the development methodology IPODLAS approach and the IPODLAS system exhibiting the main contributions of the thesis. The IPODLAS approach constitutes a general methodology suited for the development of software systems integrating applications of different domains operating on different scales collecting user requirements and application knowledge from the involved domains. The IPODLAS system is a distributed system in a heterogeneous environment which embedds legacy systems establishing and synchronizing the information exchange between them by providing a common communication model, a communication protocol, and a minimal data model,. The collected knowledge and insights and concepts of the thesis may be exploited in projects where application knowledge of the domains TSS, VR, and GIS is to be combined applying heterogeneous legacy systems.

The IPODLAS system has been developed as 'proof-of-concept'-prototype meaning that it is in its current state not ready to be applied for other advanced usages.

However, preliminary versions of the prototype have been used in the research of Price et al. (in press; submitted) (cf. subsections 6.1.1 resp. 6.1.2). To use the current IPODLAS system in a real application several preparatory work have to be considered. The current TSS legacy system RAMSES provides a number of simulation models, e.g. insect popluation modeling such as LBM or logistic regression. If the phenomenon of interest cannot be simulated using a simulation model from RAMSES, the respective algorithm must be implemented in the TSS legacy system which is may supported by the numerous auxiliary tools and libraries provided by RAMSES. The usage conditions of the other legacy systems of the IPODLAS system — i.e. VR or GIS — are analogous: if the user requests data which is not already available in the respective legacy system, this data must first be imported by the user. Due to the communication concept and the modularity of the IPODLAS system the functional extension of the current prototype is conceptually straightforward. If new functionality of existing legacy systems or new legacy systems should be made accessible for the IPODLAS system, new `IMs` have to be added to the communication protocol and the corresponding functionality of the `IpodlasClient` to wrap the functionality of the legacy system and of the `IpodlasKernel` to process the new `IMs` must be provided.

The development of the IPODLAS system regarding longer timeframes may consider the major interoperability initiatives and standards. The growing acceptance of standards and the move from stand-alone applications to network-centric approaches form the background of this thesis and provide the required means and tools to combine applications of the different domains. While the IPODLAS system establish interoperability on the data level, the major interoperability standards aims for interoperability on the API level. The *web service* approach exploits the ubiquity of the internet and provides functionality of former stand-alone applications over the internet applying standard web technology. The interoperable usage of functions provided by web services is supported by exact definitions of the interface, but these definitions are limited to syntactic aspects (Riedemann and Timm, 2003). Examples of web services of the GIS domain are given by Babu (2003), Blazek et al. (2004), and Huang et al. (2001). CORBA and HLA provides solutions to integrate compliant components potentially distributed in heterogeneous software environment by defining a communication infrastructure and common APIs. Distributed usage of applications of TSS and GIS is conducted in studies combining CORBA and HLA respectivley CORBA and OGC by D'Ambrogio et al. (2004) respectively Preston et al. (2003) and Goddard et al. (2002). However, to move forward in combining functionality of the different domains a common understanding of the phenomena of interest is required. A prerequisite to achieve this is the knowledge of the semantics and data models of the other domains. Thus, the consideration of the main *interoperability initiatives* of other domains must be subject of domain-specific interoperability initiatives to obtain domain-overarching interoperability. Examples of converging HLA and OGC are given by Bernard (2001) and Wytzisk (2003). In order to overcome the isolated nature of applications of TSS and GIS the application of interoperability initiatives of computer science is beneficial.

Appendix A

Publication list

Isenegger, D., Price, B., Wu, Y., Fischlin, A., Frei, U., Weibel, R., Allgöwer, B., 2005. IPODLAS - A software architecture for coupling temporal simulation systems, VR, and GIS. ISPRS Journal of Photogrammetry and Remote Sensing, 60(1): 34-47, http://www.sciencedirect.com/science/article/B6VF4-4HK04HB-1/2/1857645a2462ced36547c3902a1c8528.

Price, B., Isenegger, D., Allgöwer, B., Fischlin, A., in prep. The influence of orography of Larch bud moth migration at the valley scale. Oikos.

Price, B., Isenegger, D., Allgöwer, B., Fischlin, A., submitted. Spatio-temporal modelling of Larch bud moth in the European Alps: The importance of data resolution. Landscape Ecology.

Wu, Y., Price, B., Isenegger, D., Fischlin, A., Allgöwer, B., Nuesch, D., accepted. Real-time 4D visualisation of migratory insect dynamics within an integrated spatio-temporal system. Ecological Informatics.

Appendix B

Curriculum vitae

Daniel Isenegger
born June 08th, 1970, in Bern, BE, Switzerland
citizen of Hohenrain, LU, Switzerland.

Education

1983 – 1989	High school in Bülach, concluded with "Matura" exam type "B".
1992 – 1998	Studies in Geography, Sciences faculty, University of Zurich, minors in Environmental Scienes and Geology (at Swiss Federal Institute of Technology Zurich, ETHZ).
1998	MSc Thesis "Erstellung einer Klassenbibliothek räumlicher Objekte zur Unterstützung wissenschaftlich-experimenteller Programmierung", advised by Dr. B. Schneider, and Prof. R.Weibel.
1998	Diploma in Geography (dipl. geogr.), University of Zurich, Switzerland.
2002 – 2006	Dissertation at the Geographic Information Systems Division, Department of Geography, University of Zurich. Title of thesis "IPODLAS — A software architecture for coupling Temporal Simulation Systems, Virtual Reality, and Geographic Information Systems", advised by Dr. B.Allgöwer, Prof. Dr. R. Weibel, and Prof. Dr. W.Schaufelberger.

Appendix C

In this Appendix an overview over the code generated for the dissertation and the IPODLAS project is given; the complete code follows on CD. In the following diagrams showing the interaction of different software pieces is shown. The sequence of the subsections is according to their sequence in the thesis.

C.1 The cross-scale approach: Wind field generation and statistics calculation

In subsection 6.1.2 the generation of the RAMSES wind statistics for the LBM-M11 is described. Wind observation data from MeteoSwiss are employed to generate input data for the wind simulation model NUATMOS. Then, the wind direction and wind speed fields were used by the GRASS GIS to compute wind statistics fields.

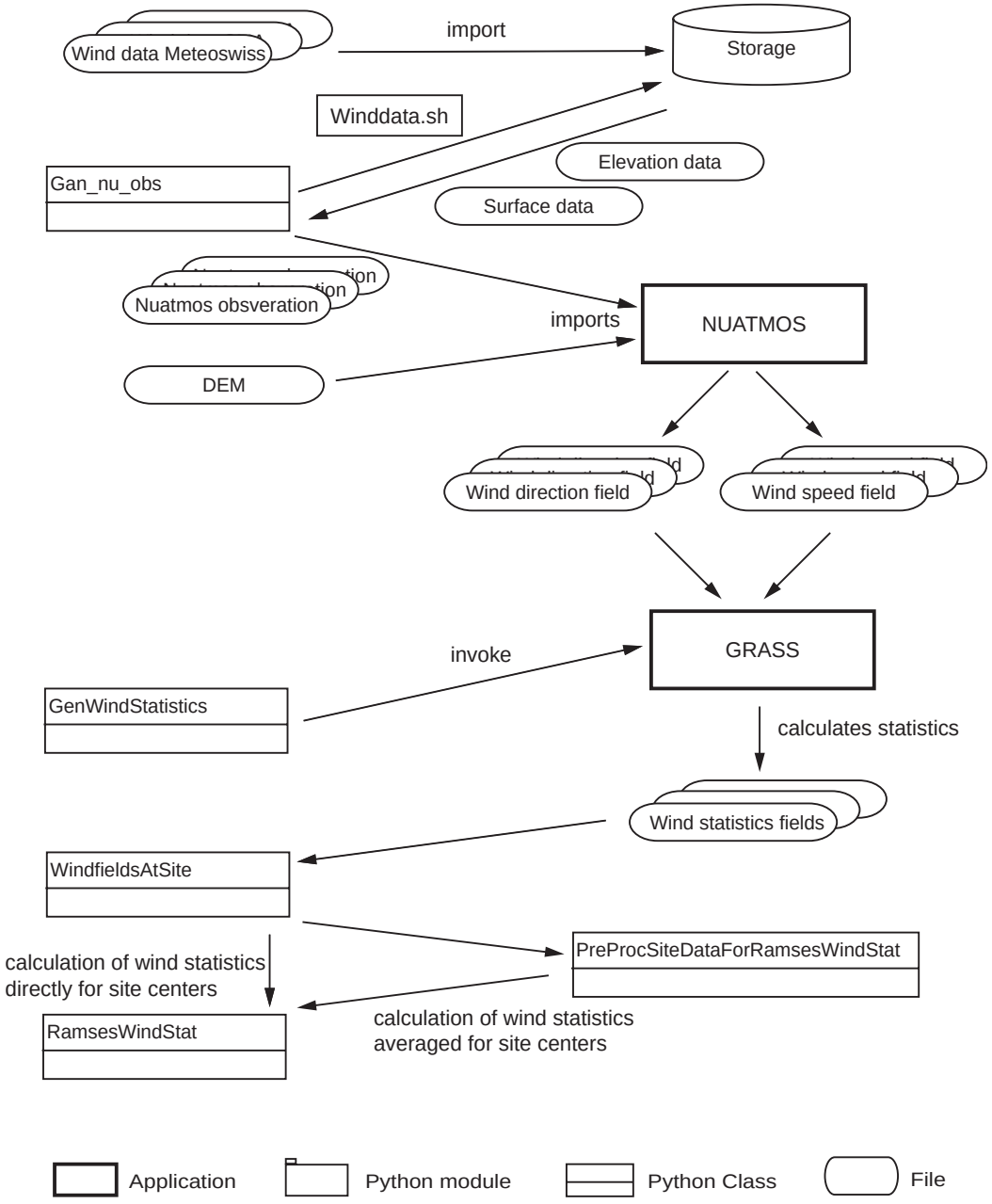

Figure C-1 Interaction of software pieces to generate RAMSES wind statistics (cf. 6.1.2).

C.2 GML 3 for describing spatiotemporal data

In subsection 6.2.1 the conversion of a temporal sequence represented in GML 2 files to a spatiotemporal GML 3 file and further to a dynamic SVG file is described. In Fig C-2 the required interaction and sequence of software pieces is illustrated.

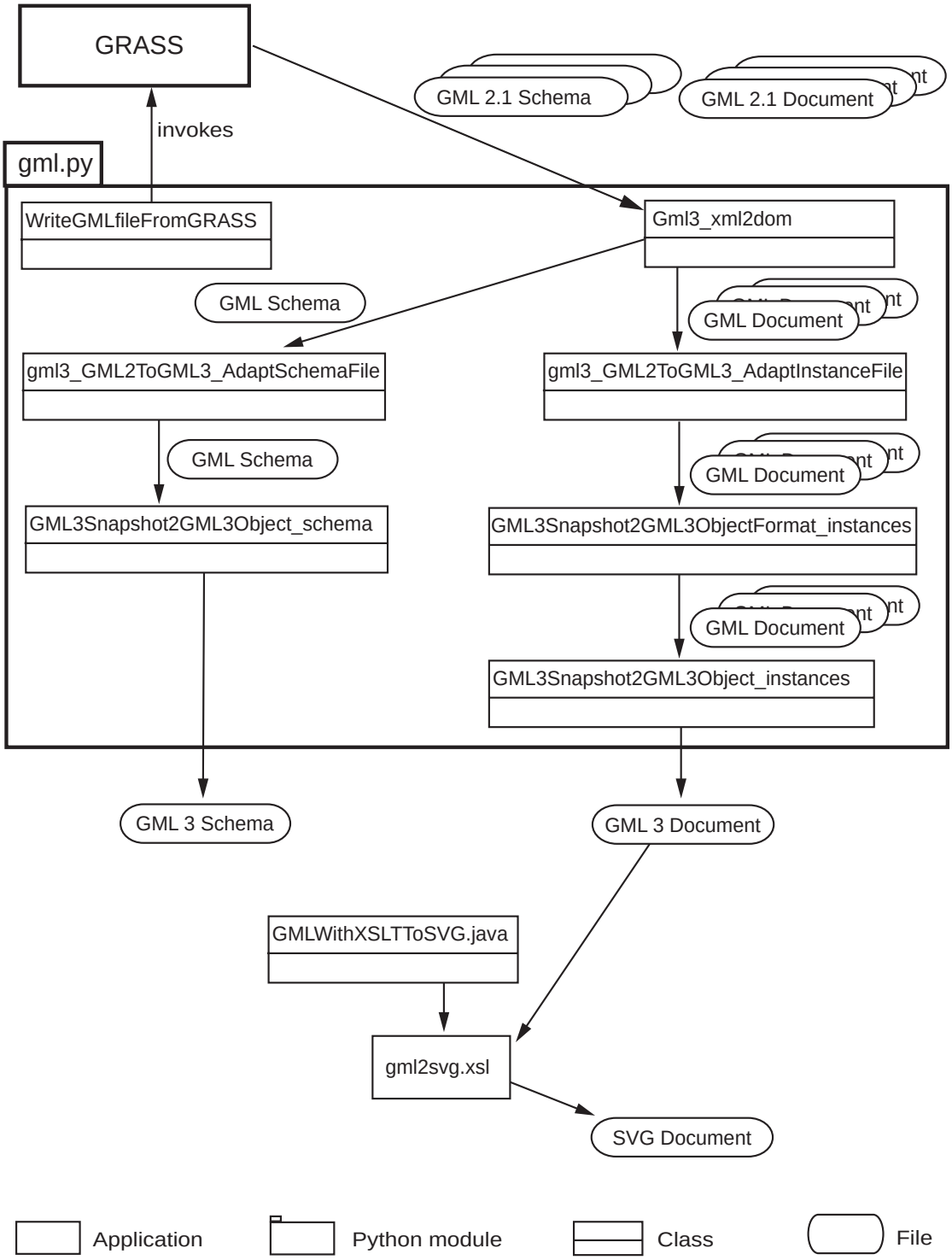

Figure C-1 Interaction of software pieces to convert a (temporal) series of GML 2 files to a spatiotemporal GML 3 file and further to a dynamic SVG file (cf. subsection 6.2.1). The Python classes of the Python module gml.py conduct the conversion from GML 2 to GML 3. The Java class then applies the XSL file to convert GML 3 to SVG.

C.3 The IPODLAS system

In Chapter 7 the final IPODLAS system is described, in particular in section 7.2 the basic classes and in section 7.3 the interaction of the basic classes. Therefore in this section only the interfaces of the classes is detailed, i.e. the main variables and fucntions of the classes, and not interaction diagrams as in the preceding sections.

The three class interfaces described in the beginning of this subsection — `IpodlasMessage`, `IpodlasKernel`, and `IpodlasClient` — constitute the core functionality of the IPODLAS system, which includes the control and the data flow. The rest of class interfaces — AccessFile, Config, GisEvent, GuiEvent, Log, Log_file, Logfile, SubSysEvent, TssEvent, VrEvent — establishing helper functionality applied by the classes of the core functionality is listed aphabetically.

C.3.1 IpodlasMessage

`IpodlasMessage` is concerned with the handling of the messages `ipodlasMessage` (cf. subsection 7.3.2) and provides the hash table, which is the data structure storing the `ipodlasMessages`. `IpodlasMessage` is derived from the TwistedMatrix class `LineOnlyReceiver`. The following interface specification in the Python module `ipodlasMessage.py` defines the class `IpodlasMessage`.

```
##############################################################################
##############################################################################
# ipodlasMessage.py
##############################################################################
# handles ipodlasMessages and provides hash table storing them
class IpodlasMessage(LineOnlyReceiver):
    #
    ####################################
    # class variables
    ####################################
    # IpodlasMessage
    #
    # class attribute for Ipodlas message handling: all instances of
    # IpodlasMessage access the same ipodlasMessage.
    # key: ID [int]
    # ipodlasMessage[0]: sender [string], e.g. GUI, GIS, TSS, VR
    # ipodlasMessage[1]: rel_id (list of related IDs) [int]
    # ipodlasMessage[2]: type (request or receipt) [string]
    # ipodlasMessage[3]: command [string]
    # ipodlasMessage[4]: evtl paramterlist
    ipodlasMessage = {}
    #
    ####################################
    # iData
    #
    # dict holding metadata about iData stored on kernel
    # key: filename
    # iDataDict[0]: topic, i.e.< LBM|WLF> [string]
    # iDataDict[1]: region, e.g. Upper Engadine[string]
    # iDataDict[2]: time range, e.g. 1951-1953 [string]
    # iDataDict[3]: content description (from iDat item <contDesc>) [string]
    # iDataDict[4]: related IMs [string]
    iDataDict = {}
    ######################################################################
    def __init__(self):
        self.id
        self.associatedSubsystem
    ######################################################################
    # sets class var self.Socket2Subsys
    def initSocket2Subsys(self):
    ######################################################################
    # load iDataDict
```

```
def loadIdataDict(self):
########################################################################
#
########################################################################
########################################################################
# getters and setters
########################################################################
# return elements of associatedSubsystem
def getAssociatedSubsystem(self):
########################################################################
# return IpodlasMessage.socket2Subsys.socketinstance (a filehandle)
# to the subsystem theSubsys
def getSocket2Subsys_instance(self,theSubsys):
########################################################################
# sets IpodlasMessage.socket2Subsys.socketinstance (a filehandle)
# to the subsystem theSubsys to the socketInstance theSocketInstance
def setSocket2Subsys_instance(self,theSubsys,theSocketInstance):
########################################################################
# return IpodlasMessage.socket2Subsys.socketAddress (a list)
# of socket to to the subsystem theSubsys
def getSocket2Subsys_address(self,theSubsys):
########################################################################
# return IpodlasMessage.socket2Subsys.socketMessage (a list)
# of socket to to the subsystem theSubsys
def getSocket2Subsys_messages2Vr(self,theSubsys):
########################################################################
# return IpodlasMessage.socket2Subsys.socketMessage (a list)
# of socket to to the subsystem theSubsys
def getSocket2Subsys_messagesFromVr(self,theSubsys):
########################################################################
# returns self.id
def getId(self):
########################################################################
# sets self.id to the value theId and the associated subsystem, e.g.
# self.id = IC_GUI and self.associatedSubsystem = GUI
def setId(self,theId):
########################################################################
# returns the list of commands
def getCmdList(self):
########################################################################
# returns the list of attributes defining the Im.cmd theCmd
# currently: def for "visuDefOlLBM" and "simWLF_withCoors"
# prm: a cmd from IM.commandList
def getImCmdDefAttrList(self,theCmd):
########################################################################
# returns the list of values (which are allowed in the associated
# attributes) defining the Im.cmd theCmd
# currently: def for "visuDefOlLBM" and "simWLF_withCoors"
# prm: a cmd from IM.commandList
def getImCmdDefValList(self,theCmd):
########################################################################
# returns the list of content Descriptions: hold in cmd-element in
# iDat-files
def getContDescList(self):
########################################################################
# return the list with the ipodlasIdentifiers
def getIpodlasIdList(self):
########################################################################
# return the list with iData topics
def getIdataTopicList(self):
    """returns the list with iData topics."""
    return IpodlasMessage.iDataDopicList
########################################################################
# return the list with iData regions
def getIdataRegionList(self):
########################################################################
# returns ipodlasStorage path
def getIpodlasStoragePath(self):
```

```
########################################################################
# return filenameList, which holds serialized iDataDict and "normal"
# filename
def getIdataDictFilenameList(self):
########################################################################
# set iDataDict: adds an entry in iDataDict with the key theFilename
# and the value theTopic,theRegion,theTimerange,theContDesc,theRelImList
def setIdataDict(self, theFilename,theTopic,theRegion,theTimerange,\
                 theContDesc,theRelImList):
########################################################################
# returns value of iDataDictloaded: <"Yes"|"No">
def getIDataDictloaded(self):
########################################################################
# proviorisch: returns iDataDict
def getIdataDict(self):
########################################################################
# list with cmds from IM.commandList, which may trigger generation of
# iData  return iData_cmdList list
def getIdata_cmdList(self):
########################################################################
# get the item theItemNumber from iDataDict[theKey]
# prm:
# - theKey is a filename
# - theItemNumber: number of item in dict, i.e. 0 - 5
# returns: content description [string]
def getIdataDictItem(self,theKey,theItemNumber):
########################################################################
# get list of name of files on VR, which are used to exchange data with
# VR
def getFilenameOnVrList(self):
#
########################################################################
########################################################################
# general functions                                                    #
########################################################################
# invoke the responsible IK theIK to send data thedata to its
# associated IC
# prms:
# - theData: a string containing the data to be sent to respective IC
# - theIK: the IK (must be retrieved trough IM.getIpodlasIdList()
def sendData2Ic(self,theData,theIK):
########################################################################
# compress the file theFile and return name of compressed file
# using ZIP
# prm:
# - input: name of file theFile
# - returns: name of compressed file
def compressFile2Zipfile(self,theFile):
########################################################################
# decompress the file theFile and returns the bytestream
# using ZIP
# prm:
# - input: name of file theFile
def decompressFile2ByteStream(self,theFile):
########################################################################
########################################################################
# IpodlasMessage part: routines dealing with IMs
########################################################################
# generates a unique Ipodlas message ID for a Ipodlas message
def genIm_id(self):
########################################################################
# if IPODLAS message is a new one (i.e. if no ID exists):
# generate Ipodlas message id, generate a new entry in class var
# ipodlasMessage and return the generated message id im_id
def newIpodlasMessage(self,theImList):
########################################################################
# inserts values of Ipodlas message (contained in theImList) into class
# var dict ipodlasMessage and return the message id
```

```
def setIpodlasMessage(self,theImList):
#####################################################################
# returns the ipodlasMessage with the ID theImID as a list
def getIpodlasMessage(self,theImID):
#####################################################################
# this function may only invoked from IK (since it creates a new im_d):
# generates a receipt for the IPODLAS message, adds it to IpodlasMessage
# with the ID theImID and return im_id of the receipt
def genReceiptInKernel(self,theImID):
#####################################################################
# this function may only invoked from IC (does not create a new im_id):
# generates a receipt for the IPODLAS message theImID, and return
# the im_list
def genReceiptInClient(self,theImID):
#####################################################################
# generates an string containg the XML code of an ipodlasMessage
# input prm: ID theImID of an ipodlasMessage in self.ipodlasMessage
# return prm: impodlasMesasge as XML in a string
def genXML(self,theImID):
#####################################################################
# generates an string containg the XML code of an ipodlasMessage
# input prm: ID theImID of an ipodlasMessage in self.ipodlasMessage
# return prm: impodlasMesasge as XML in a string in prettyPrint
# (i.e.with lineBreaks)
def genXML_prettyPrint(self,theImID):
#####################################################################
# generates an string containg XML code of an ipodlasMessage
# input prm: the list theImList with the prms of an ipodlasMessage
# return prm: impodlasMesasge as XML in a string
def genXMLFromList(self,theImList):
#####################################################################
# generates an string containg XML code of an ipodlasMessage in
# prettyPrint (i.e. with newline and tabs)
# input prm: the list theImList with the prms of an ipodlasMessage
# return prm: impodlasMesasge as XML in a string
def genXMLFromList_prettyPrint(self,theImList):
#####################################################################
# parses the string theData if it is an 'im',an 'iData' or a 'zip'
# returns a string: im or iData
def parseData(self,theData):
#####################################################################
# parses the string theData and extracts the values
# return value: list containing the values of the theData if theData
# is of type ipodlasMessage
def parseIpodlasMessage(self,theData):
#####################################################################
# parses cmd-string, which is located in IM in the cmd-item
# input prm: id of IM
# returns list of indices
# ([identify_index,display_index,visu_index,read_index])
#      -1: if not in cmd, > -1 otherwise
def parseCmd_fromId(self,theImID):
#####################################################################
# parses cmd-string, which is located in IM in the cmd-item
# input prm: IM list
# returns list of indices
# ([identify_index,display_index,visu_index,read_index])
#      -1: if not in cmd, > -1 otherwise
def parseCmd_fromList(self,theImList):
#####################################################################
# parse the command string theCmdString
def parseCmdElements(self,theCmdString):
#####################################################################
# parse the command string theCmdString into elements separated by ' ',
# which is e.g.
# 'readDataFile id=TSS
# name=TSS_females_migrating_1_1949_1951.txt time=1949-1957 site=1,20
# winterEggMort=0.65'
```

```
#
def parseCmdReadDataFile(self,theCmdString):
######################################################################
# parse the command string theCmdString into elements separated by ' ',
# which is e.g.
# 'simuLBM time=1949-1959 site=1,20 winterEggMort=0.65'
def parseCmdSimulLBM(self,theCmdString):
######################################################################
# parse the command string theCmdString into elements separated by ' ',
# which is e.g.
# 'getGGInfo coors=(123456.00,123456.00,123456.00)'
def parseCmdGetGGInfo(self,theCmdString):

######################################################################
######################################################################
# iData part: routines dealing with iData                            #
######################################################################
# writes beginning of an iData string
# input prms:
# - theFileName: the filename
# - theDescOfCont: description of file content
# - rel_im: a list of ids of related IMs, i.e. IMs that triggered
#           generation of the iData
# returns the begining of an iData string in pretty print
def initIdataString(self,theFileName,theDescOfCont,theRelImList):
######################################################################
# parses iData theIdata for the elements fileName, content desc,
# ids of related IMs
# input prm: the iData string
# return prm: iDataList
# iDataList[0]: fileName
# iDataList[1]: contDesc
# iDataList[2]: rel_im list
def parseIdata(self,theIdata):
######################################################################
# parse time range from idat.content desc
# returns: time range, e.g. 1951-1952
def getTimeRangeFromContDesc(self,theContDesc):
######################################################################
# queries if iData for LBM exists for request with id theImID
def existIdataForLbm(self,theImID):
######################################################################
# searches iDataDict.contDesc for the string theSearchString
def searchIdataDictContDesc(self,theSearchString):
######################################################################
######################################################################
# Ramses Result part: routines dealing with Ramses result files  #
######################################################################
# Generates for a Ramses result file (as a string) from old header
# the new header with the new start and endyear
# returns: the new header as string
def genNewHeader(self,theOriginalHeaderString,theStartYear,theEndYear):
######################################################################
# For visualization: in a original Ramses result file
# (as the string theDataString)
# the 20 defoliation values of the year theStartYear are parsed
# return: defoliation value string (contains 20 defol values)
def genDefoliationValueString(self,theDataString,theStartYear):
######################################################################
# For visualization: in a Ramses result file in IpodlasStorage
# (read in as as the string theDataString)
# diff to genDefoliationValueString(): here an original ramses result
# file is used (this has e.g. no header)
# the 20 defoliation values of the year theYear are parsed
# returns: defoliation value string (contains 20 defol values)
def genDefolFromRamsesFilesInIpodlasStorage(self,theDataString,theYear):
######################################################################
# parses the data values from the Ramses data string theDataString
```

```
    # returns: list with defoliation values
    def parseDefolValuesFromRamsesDataString(self,theDataString):
    ######################################################################
    # cut out the requested years (startYear to endYear)
    # from string theDataString (representing a Ramses result file)
    # return: the datastring cut out according to requested time range
    # (theStartYear to the theEndYear)
    def cutOutRequestedYearsFromData
                          (self,theDataString,theStartYear,theEndYear):
    ######################################################################
    # not used at the moment: when parsing the ramses result file
    # several strange effects happened
    #returns list of positions of occurences of the substring theSubstr
    # in the string theStr
    # position: position of first char of substr
    def getOccurencesOfSubstrInStr(self,theStr,theSubstr):
######################################################################
if __name__ == "__main__":
    # init
    im = IpodlasMessage()
```

C.3.2 The IpodlasKernel

`IpodlasKernel` (cf. section 7.3) is deployed on the kernel establishing the communication with the `IpodlasClient` instances on the subsystems. To inherit the functionality, it is derived from `IpodlasMessage`. `IpodlasKernelFactory` is used to instantiate and manage `IpodlasKernel`. Both classes are defined within the following Python module `ipodlasKernel.py`.

```
##############################################################################
##############################################################################
# ipodlasKernel.py
##############################################################################
class IpodlasKernel(IpodlasMessage):
    ######################################################################
    # initialize variables
    def __init__(self):
        self.lg # logfile
    ######################################################################
    # called by LineOnlyReceiver.dataReceived()
    def lineReceived(self, data):
    ######################################################################
    def getId(self):
    ######################################################################
    # called, when a client first connects
    def connectionMade(self):
    ######################################################################
    # removes client from list self.clients in factory
    def connectionLost(self, reason):
    ######################################################################
    # kernel function: main control flow control class
    def kernelControl(self,theData):
    ######################################################################
    # processes requests from clients:
    def processRequest(self,theImID):
    ######################################################################
    # processes receipts from clients:
    # prms: IMlist
    def processReceipt(self,theImList):
    ######################################################################
    # processes Idata sent from ICs; this is iData generated by simulation
    # prms: the theIdataList describes the iData,theData is the data
    def processIdata(self,theIdataList,theData):
    ######################################################################
    # handle the iData stream (desc by theIdataList)
    def handleIdata(self,theIdataList,theData):
```

```
    ########################################################################
    # writes the xml-string iData to file in the ipodlasStorage
    # and writes the metadata dict iDataDict  to a file and serializes
    # it also to file
    # prms: the theIdataList describes the iData,theData is the data
    def storeIdata(self,theIdataList,theData):
    ########################################################################
    # check if iData for current request with id theImID is available
    def checkAvailabilityOfIdata(self,theImID):
    ########################################################################
    # since file theNameOfIdatfile contains the data requested
    # IK.simulateLbmUE()/simulateWlf() is not done, i.e.
    # - the file with the requested data theFile is read and
    # parsed using  IM.parseIdata()
    # - the function-chain is resumed by calling processIdata()
    def skipSimu(self,theNameOfIdatfile):
    ########################################################################
    # generates a request (display a raster) in XML for the request with
    # the ID
    # theImID and sends it to the appropriate IK-instance
    def displayRaster(self,theImID):
    ########################################################################
    # generates a request (read file from subsys using files) in XML for
    # the request with the ID
    # theImID and sends it to the appropriate IK-instance
    def visualizeLbmUe_usingFiles(self,theImID):
    ########################################################################
    # generates a request (read file from subsys using sockets) in XML
    # for the request with the ID
    # theImID and sends it to the appropriate IK-instance
    def visualizeLbmUe_usingSockets(self,theImID):
    ########################################################################
    # generates a request (read file from subsys using mixed approach, i.e.
    # using sockets and files) in XML
    # for the request with the ID  theImID and sends it to the
    # appropriate IK-instance
    def visualizeLbmUe_usingMixed(self,theImID):
    ########################################################################
    # - send the iData file to IK_VR
    # - sends a request (an IM) to IK_VR and triggers to visualize LBM
    # defoliation read from a iData file, whose name can be
    # retrieved in the cmd item of the IM theImID
    #
    # IC.visuDefolLBMAndGetCoor():receives a pair of coordinates
    def visuDefolLBMAndGetCoor(self,theImID):
    ########################################################################
    # generates a request (which is read from file by the IC on respective
    # subsys)
    # in XML for the request with the ID theImID and sends it to the
    # appropriate IK-instance
    def stopVisualize(self,theImID):
    ########################################################################
    # generates a request (read file from local subsystem) in XML for the
    # request with the ID theImID and sends it to the appropriate IK-
    # instance
    def readFile(self,theImID):
    ########################################################################
    # generates a request (read file from local subsystem) in XML for the
    # request with the ID theImID and sends it to the appropriate IK-
    # instance
    def simulateLbmUE(self,theImID):
    ########################################################################
    # generates a request (read file from local subsystem) in XML for the
    # request with the ID theImID and sends it to the appropriate IK-
    # instance
    def readDataFile(self,theImID):
    ########################################################################
    # catching receipt of writing iDat file to VR from IC_VR: this release
```

```
    # an IM
    # requesting visualization of this data file
    # prm: an IM as list
    def processWrittenIdataOnVr(self,theImList):
    ###########################################################################
    # get an IM requesting geographic information. Invokes IC_GIS to query
    # geographic information using the coors in the cmd-item
    # the command is e.g.
    # "getGGInfo coors=(782343.289063,148205.166016,1823.127930)"
    def getGGInfo(self,theImID):
    ###########################################################################
    # sends the IM requesting displaying text in GUI to IC_GUI, the text
    # is in the cmd-part of the IM, and is e.g.
    # 'dispInGUI (782343.28,148205.166016)  is located in site 19'
    def dispGgInfoInGui(self,theImID):
    ###########################################################################
    # processes ramses result file which are sent to VR
    def processRamsesResultForVr(self,theIdataList,theData):
    ###########################################################################
    # processes ramses result file which are sent to GUI
    def processRamsesResultForGUI(self,theIdataList,theData):
    ###########################################################################
    # processes simuWlf request
    # checks if requested LBM data is available in ipodlasStorage
    # - if no: request simulation of LBM in this period of user: send a
    # message to be displayed on GUI
    #- if yes: parse the respective iData file for the defol values and
    # send simuWLF request plus an iData file with the defolvalues to VR
    # since only defoliation values are required
    def simuWlf(self,theImID):
    ###########################################################################
    # processes simWLF_withCoors request
    #
    def handleSimWlf_withCoors(self,theImID):
    ###########################################################################
    # controls simulation of wildland fire in grass
    #
    def simWlfInGrass(self,theImCmd):
    ###########################################################################
    # reads the file with name theNameOfIdatfile from IpodlasStorage
    # prms:
    # theFileName: filename of file
    # returns file as a string
    def readFileFromIpodlasStorage(self,theNameOfIdatfile):
    ###########################################################################
    # write the string theData to the file theFilename on IpodlasStorage
    # prms:
    # theFileName: filename of file
    # returns file as a string
    def writeFileToIpodlasStorage(self,theData,theFilename):
###############################################################################
#
###############################################################################
# instantiated only once
class IpodlasKernelFactory(Factory):
    # IpodlasKernel is a indirect child of protocol
    protocol = IpodlasKernel
    ###########################################################################
    def __init__(self):
       self.clients
       self.clientsAtSubsys
       self.lg # logfile
    ###########################################################################
    # adds an entry in dict clientForComp for each instance theInstance of
    # an ipodlasKernel
    # prm: identifier theInstanceIdentifier (i.e. IC_GUI,IC_GIS,IC_VR,
    # IC_TSS,...)
    # key: theInstanceIdentifier (i.e. IK_IC_GUI,IK_IC_GIS,IK_IC_VR,
```

```
    # IK_IC_TSS,...)
    # clientForComp[0]: theInstance (an instance of an ipodlasKernel)
    def identifyInstance(self,theInstanceIdentifier,theInstance):
    ########################################################################
    # when IC- or SSC-instance is deleted, removes the associated IK
    # instance from the dict clientAtSubsys
    def removeInstance(self,theKey):
    ########################################################################
    def addClient(self, newclient):
    ########################################################################
    def delClient(self, client,theId):
    ########################################################################
    def sendAll(self, message):
    ########################################################################
    # returns instance of ipodlasKernel, which communicates with client
    # theClient
    # return value: reference to instance
    def getKernelInstanceForClient(self,theClient):
##############################################################################
if __name__ == "__main__":
    # the listening socket is set up
    reactor.listenTCP(51423, IpodlasKernelFactory())
    # main loop
    reactor.run()
```

C.3.3 The IpodlasClient

`Instances of IpodlasClient` (cf. section 7.3) is deployed on the subsystems establishing the communication with the associated `IpodlasKernel` instances on the kernel. To inherit the functionality, it is derived from `IpodlasMessage`. `IpodlasClientFactory` is used to instantiate and manage `IpodlasClient`. Both classes are defined within the following Python module `ipodlasClient.py`.

```
##############################################################################
##############################################################################
# ipodlasClient.py
##############################################################################
# establishes connection to IpodlasKernel over sockets and to subsystems via
# file exchange
class IpodlasClient(IpodlasMessage):
    ########################################################################
    # initialize variables
    def __init__(self,theId):
        self.lg # logfile
    ########################################################################
    # overwrites the lineReceived from base class
    def lineReceived(self, data):
    ########################################################################
    def clientControl(self, theData):
    ########################################################################
    # processes Idata sent from IKs:
    # prms: the theIdataList describes the iData,theData is the data
    def processIdata(self,theIdataList,theData):
    ########################################################################
    # processes zippedData sent from IKs:
    # prms: theData is the zipped data
    def processZippedData(self,theData):
    ########################################################################
    # identifies client: i.e. sends identifying receipt to ipodlasKernel
    # prm: the id of the IM which requests to identify
    def identify(self,theImID):
   ########################################################################
    # read (information exchange) file of the associated subsystem,
    # e.g. infoEx_fromGUI.txt
    def readFile(self):
    ########################################################################
```

```
    # read data file of the associated subsystem
    # prm: IM id
    def readDataFile(self,theImID):
    ###################################################################
    # since only files of the size < 16384 chars can be transferred between
    # IC and IK:
    # - user requests have result in limited filesize , i.e to be limited
    # to 2 years
    # - the requested years from the existing data file,
    # e.g. TSS_females_migrating_1.txt have to cut out and a adapted
    # header have to be generated
    # input prms:
    # returns: name of generated file newFileName
    def generateRequestedTssResultfile(self,theRamsesResultfile,\
                                theStartYear,theEndYear):
    ###################################################################
    # invokes GIS to display the raster cmd, using IpodlasProcess to
    # call a process on the machine
    def displayRaster(self, theCmd):
    ###################################################################
    # handles the successfull call of the display of the raster theRaster
    def handleDisplayRaster(self,theRaster):
    ###################################################################
    # handles the failed call of the display of the file theFoile
    # and its return object theFailure
    def handleDisplayRasterFailure(self,theFailure):
    ###################################################################
    # invokes VR (using a mixed approach, i.e. socket and files) to display
    # the LBM in UE
    # prm: id of the respective IM
    def visualizeLbmUe_usingMixed(self, theImID):
    ###################################################################
    # process return messages theData from VR
    # - generates a receipt when receiving 'VTP', that visu is done
    # - generates a request when receiving 'getGGInfo': in order to query
    # data from GIS
    # imput prm: IM, theData, which is
    # e.g. getGGInfo coors=(782343.28,148205.166016,1823.127930)
    # return prm: vrProcessList:
    # vrProcessList[0]: <break|notBreak>
    # vrProcessList[1]: im as XML string
    def processMessageFromVR(self,theImID,theData):
    ###################################################################
    # invokes VR to display the LBM in UE
    # prm: id of the respective IM
    def visuDefolLBMAndGetCoor(self, theImID):
    ###################################################################
    # generates a request to simulate WLF
    # prm:
    # - theVrResponse: a string (def in IM.socket2Subsys), e.g.
    # "selecetedIgnitionPoint coors=(782343.28,148205.166016,1823.127930)"
    # - theImID: id of IM which triggers invocation of req2SimuWlf()
    def req2SimuWlf(self,theVrResponse,theImID):
    ###################################################################
    # invokes VR (via socket) to stop visualize the LBM in VR
    # prm: id of the respective IM
    def stopVisualize(self,theImID):
    ###################################################################
    # invokes the generation of a a visual receipt (e.g. an alert Box),
    # which can be displayed by the GUI
    # therefore: this fctn can only be executed when GUI is addressable,
    # i.e. at the moment on the mac platform
    def genVisualReceiptOnGuiPlf(self,theImID):
    ###################################################################
    # invokes the TSS application to simulate
    def simulateLbmUE(self,theImID):
    ###################################################################
    # parse years from string theString
```

```
# return the 2 year in a list
def getYears(self,thetimeString):
#########################################################################
# processes results from TSS for VR
def processTssResultsForVr(self,theIdataList,theData):
#########################################################################
# processes results from TSS for GUI
def processTssResultsForGUI(self,theIdataList,theData):
#########################################################################
# get an IM requesting geographic information. Invokes the associated
# GIS to query
# geographic information using the coors in the cmd-item
# the command is e.g.
# "getGGInfo coors=(782343.289063,148205.166016,1823.127930)
def getGGInfo(self,theImID):
#########################################################################
# writes text to the file infoEx_toGUI and invokes an applescript
# to dispaly the filecontent in an alert-box.
# the text to write to the file is in the cmd-part of the IM, and is
# e.g.
# 'dispInGUI (782343.28,148205.166016)  is located in site 19'
def dispGgInfoInGui(self,theImID):
#########################################################################
# reads from file theIdataFilename text to disply it in an alert-box.
# This must be done only on the plf GUI (a mac plf),since applescript
# are used
def dispRamsesResultInGui(self,theIdataFilename):
#########################################################################
# processes simWLF_withCoors request. this is invoked from
# IC_VR.req2SimuWlf()
#
def simWlf_withCoors(self,theImID):
#########################################################################
# controls simulation of wildland fire in grass
# prm:
# - theImCmd:
# e.g.
# imCmd = 'simWLF_withCoors time=1976 moistureLive=accLbmDef
#         ignitionPoint=(787671.28,152305.166016,1823.127930)
#         filename=defolValues_1953'
#
# ge.processGisEvent(): generates a ascii-raster named "wlfSpread.asc"
# and returns: name of WLF spread file
def simWlfInGrass(self,theImCmd):
#########################################################################
# processes WLF-simulation results on VR. The bytestream theZippedData
# must be written to file from where it can be decompressed
# prm:
# theZippedData: zipped asci-raster
def handleWlfSimuOnVr(self,theZippedData):
#########################################################################
# check if current platform a mac platform is
# return macCheck: <isMac|isNoMac>
def macCheck(self):
#########################################################################
# invoke the appleScript theAppleScript on a mac
# the appleScript: must be located in the path ipodlasPath
def invokeAppleScript(self,theAppleScript):
#########################################################################
# writes the string theData (desc by the metadata theIdataList)
# to the local disk
def writeIdat2LocalDisk(self,theIdataList,theData):
#########################################################################
# establish a socket connection to a socket server on the subsystem
# theSubSys. The message theMessage is sent and waits for an answer
# establish as client a socket to socket server on subsys theSubSys
# client sends message theMessage
# Then client listens on socket for confirmation
```

```
    # theConfirmationFromSubsys
    # prms:
    # - theSubSys: currently VR
    # - theMessage: the message sent from here to socket server
    # - theConfirmationFromSubsys: the confirmation from the socket server
    # (used to parse against)
    # returns: the confirmation from socket server
    def socket2SocketServer (self,theSubSys,theMessage,
                                                    theConfirmationFromSubsys):
##############################################################################
#
##############################################################################

# IpodlasClientFactory is derived from ReconnectingClientFactory, which
# tries to reconnect if connection is lost
class IpodlasClientFactory(ReconnectingClientFactory):
    ############################################################################
    # creates protocol and receives events relating to connection state
    def __init__(self,theId):
        self.id
    ############################################################################
    def startedConnecting(self, connector):
    ############################################################################
    def buildProtocol(self, addr):
    ############################################################################
    def clientConnectionLost(self, connector, reason):

    ############################################################################
    # if connection could not be established
    def clientConnectionFailed(self, connector, reason):
        print 'Connection failed. Reason:', reason
######################################################################
# global routine to start up vrEvent
def startUpVrEvent():
######################################################################
if __name__ == "__main__":
    # connect to (host,port) and start reactor
    from twisted.internet import reactor
    reactor.connectTCP("130.60.176.97",51423,IpodlasClientFactory(ID))
    reactor.run()
```

C.3.4 AccessFile

The class `AccessFile` provides the functionality concerning file handling. It is defined in the Python module `accessFile.py`.

```
##############################################################################
##############################################################################
# accessFile.py
##############################################################################
# accesses a file on the disk; all file handling is done with relative
# filenames the absolute name (i.e. adding the path) is done by in
# openFileWithSemaphore() by placing all files
# in the current dir
class AccessFile:
    ############################################################################
    # init access file with the ID of the subsystem and the path thePath
    # where the files this class accesses are located
    def __init__(self,theID):
         self.id
         self.currentDir
         self.lg # logfile
    ############################################################################
    # parse subsystem from id
    def init_subsys(self, theId):
    ############################################################################
    # initialize information exchange dict self.infoExchange for the
```

```
    # associated subsystem
    def init_infoExchangeFiles(self):
    ###########################################################################
    # writes the IM as an xml string theImAsXML to an InfoExchangeFile:
    # if fromOrTo == "from": then e.g. in infoEx_fromGUI
    # if fromOrTo == "to": then e.g. in infoEx_toGUI
    def writeIm2InfoExchangeFile(self,theImAsXML,fromOrTo):
    ###########################################################################
    # reads the IM as an xml string theImAsXML from an InfoExchangeFile,
    # e.g. from infoEx_toGUI.txt
    # returns the IM as xmlstring
    def readImFromInfoExchangeFile(self,fromOrTo):
    ###########################################################################
    # reads data from the datafile theDatafile, e.g.females_migrating_1.txt
    # returns the data as one iData string
    def readIdataFromDataFile(self,theDatafile):
    ###########################################################################
    # writes theString to (information exchange) file theFile
    # if no semaphore theSemaphore exists
    # theSemaphore: can be set as "NONE"
    def writeFile(self, theFile,theSemaphore,theString):
    ###########################################################################
    # read file theFile if no semaphore theSemaphore exists
    # return: file content as string fileContent
    def readFile(self,theFile,theSemaphore):
    ###########################################################################
    # opens the file theFile (only relative filenames) with the mode
    # theAccessMode, but checks first if
    # no associated semaphore file theSemaphore exists
    # if yes, then file access is delayed and tried again later
    # if no: the associated semaphore file is generated
    # return prm: filehandle of the opened file or "NONE"
    def openFileWithSemaphore(self,theAccessMode,theFile,theSemaphoreFile):
    ###########################################################################
    # close the file accessed with the filehandle theInfoExchange_fh and
    # removes
    # the semaphore file accessed with the filehandle theSemaphore_fh
    def closeFileWithSemaphore(self,theSemaphore_fh,theInfoExchange_fh):
###########################################################################
if __name__ == "__main__":
    af = AccessFile(id)
```

C.3.5 Config

In the class `Config` configuration information of the IPODLAS system is defined, i.e. file names and paths. `Config` is defined in the Python module `config.py`.

```
###############################################################################
###############################################################################
# config.py
###############################################################################
# configuration class: in this files paths, filenames, etc are defined
# Those configurations should be used within the ipodlas members
class Config:
    ###########################################################################
    def __init__(self):
        self.platformID =         # platform names
        #
        # dict holding the parameter for the respective guiRequest
        # - key: a guiRequest: i.e. ["stopLbm","runLbm","runWlf"]
        #                           (possible requests fired from GUI)
        # - list
        #      - [0]: guiRequestListLength (number of parameters
        #                                   beloning to an item of
        #                                   guiRequest)
        #      - [1]: attributes used in IM.<cmd>
        #                                (beloning to an item of guiRequest)
```

```
        #      - [2]: values fired from GUI
        #      - [3]: values used in used in IM.<cmd>
        self.guiRequestDict =
        #
        self.ipodlasPath = ""
    ##########################################################################
    # init fctns
    ##########################################################################
    # init platform
    def initPlatformDict(self):
    ##########################################################################
    # getter and setter
    ##########################################################################
    # determine the ipodlasPath depending on the platform
    def setIpodlasPath(self):
    ##############################################################################
    # returns ipodlasPath
    def getIpodlasPath(self):
    ##############################################################################
    # get platformNameList
    def getPlatformNameList(self):
        return self.platformNameList
    ##############################################################################
    # queries the current platform e.g.
    def getCurrentPlatformName(self):
    ##############################################################################
    # returns PlatformDict
    def getPlatformDict(self):
    ##############################################################################
    # returns guiRequest, e.g. ['stopLbm','runLbm','runWlf']
    def getGuiRequestList(self):
    ##############################################################################
    # returns length of guiRequestList of guiRequest theGuiReq
    def getGuiRequestListLength(self,theGuiReq):
        return self.guiRequestDict[theGuiReq][0]
    ##############################################################################
    # returns associated attributeList of guiRequest theGuiReq,
    # used in IM.<cmd>, e.g. ['time','site','winterEggMort']
    def getGuiRequestAttrsList(self,theGuiReq):
    ##############################################################################
    # returns associated argumentList (fired from GUI) of guiRequest
    # theGuiReq sent from GUI to guiEvent.py, e.g. ['default from
    # IPODLAS','according to LBM defoliation','set in VR']
    def getGuiRequestArgsList(self,theGuiReq):
    ##############################################################################
    # returns associated valueList for attributeList of guiRequest
    # theGuiReq, used in IM.<cmd>,
    # e.g. ['defFromIpodlas','accLbmDef','setInVr']
    def getGuiRequestValList(self,theGuiReq):
    ##############################################################################
    # returns ramsesResultFilenameList
    def getRamsesResultFilenameList(self):
    ##############################################################################
    # returns list containig names of information exchange files
    def getMacInfoExchangefileList(self):
    ##############################################################################
    # returns winterEggMortBorder
    def getWinterEggMortBorder(self):
##############################################################################
if __name__ == "__main__":
    #
    # init Config
    conf = Config()
```

C.3.6 GisEvent

The class `GisEvent` is responsible for the communication with the GIS legacy system. It receives requests from the `IpodlasClient` instance deployed on the GIS subsystem and invokes the GIS legacy system to handle the request and propagates the results back to the local `IpodlasClient` instance. `GisEvent` is defined in the Python module `gisEvent.py`.

```
##############################################################################
##############################################################################
# gisEvent.py
##############################################################################
# handles GIS events:
# - catches GIS requests sent from the IpodlasClient
# - propagates GIS results back to the IpodlasClient
class GisEvent:
    ########################################################################
    def __init__(self):
        self.id
        self.gisDataPath
        self.lg = LoggingText()# init logfile
        #
        ########################################
        # GRASS
        ########################################
        # grassRegionDict: contains information for the available regions
        # key: region [string]
        # [0]: defining region [string], i.e. defines extent of region
        self.grassRegionDict =
        #
        # grassDBDict: contains information for the available DBs
        self.grassDbDict = {\
            "postgres" : 'db.connect driver=pg
             database="dbname=mydb,user=disen"'}
        #
        # key: region [string]
        # UpperEngadine: defining region, i.e. defines extent of region
        # site_id: defines areas of sites, contains values 1-20 an NULL
        # moisture_live: def live moisture content of region of
        # UpperEngadine ( 120 for grass, 130 for forest)
        self.grassRegionDict
        #
        # contains coors of fire orgin, e.g. '1|785775|151675|fireOrigin'
        # e.g. used as input file for v.in.ascii
        self.fireOriginFile         #
        # ESRI ascii raster: output of WLF spread
        self.wlfSpread = "wlfSpread.asc"
        #
        # the dict grassRrosData defines the input and output data of
        # the command r.ros
        self.grassRrosData = {\
            "fuel","moisture_1h","moisture_live","velocity","direction" :,
            "slope","aspect","elevation","output": ["my_ros"]
        #
        # the dict grassRspreadData defines the input and output data of
        # the command r.spread
        self.grassRspreadData = {\
            "max" :,"base","dir" : ["my_ros.maxdir"],"w_speed",
            "f_mois","start","lag","backdrop",output","x_output","y_output"
        # simulation results from previous simulations
        self.prevSimuList = ["my_ros.max","my_ros.base","my_ros.maxdir",\
                       "my_ros.spotdist","my_spread","my_spread.x",\
                       "my_spread.y","my_path","liveMoisAccDefol"]
        #
        # the dict liveMoistureAndDefol shows the mapping of the defoliation
        # values of the sites to the associated live moisture content:
```

```
    # undefoliated: < 10 % (defoliation in site)
    # little: >= 10% and < 30%
    # moderate: >= 30% and < 60%
    # much: >= 60% and < 90%
    # all: >= 90%
    self.liveMoistureAndDefol
########################################################################
# getters and setters
########################################################################
# init region in GRASS
# prm:
# - theRegion: must be read from grassRegionDict
def setGrassRegion(self,theRegion):
########################################################################
# init DB, which uses GRASS
# prm:
# - theDB: must be a key from self.grassDBDict, currently only
# 'postgres'
# is implemented. i.e. postgres must be up and running
def setGrassDB(self,theDB):
########################################################################
# set Grass monitor
def setGrassMon(self):
########################################################################
########################################################################
# gisEvents
########################################################################
# processes a GIS event: GisEvent is invoked from Ipodlas to process
# requests, which can be handled by the current GIS
# prms:
# - input: e.g. imCmd = 'simWLF_withCoors time=1976
#               moistureLive=accLbmDef
#               ignitionPoint=(787671.28,152305.166016,1823.127930)
#         filename=TSS_females_migrating_1_1976_1977_20051014103256.txt'
# - returns: ascii-raster wlfSpread_asc
def processGisEvent(self,theImCmd):

########################################################################
# retrieves the value from the raster theRaster at (theX,theY)
# returns: the value
def getValueFromRaster(self,theRaster,theX,theY):
########################################################################
# removes from Grass DB the file theFile
# e.g. old raster from previous simulations
def gRemove(self,theFile):
########################################################################
# Converts the binary GRASS vector map layer theVectorLayer into a GRASS
# raster map layer theRasterLayer
def v2rast(self,theVectorLayer,theRasterLayer):
########################################################################
# Converts a (grass)raster map layer theGrassRaster into an
# ESRI ARCGRID (ascii) raster theAsciiRaster
def rOutArc(self,theGrassRaster,theAsciiRaster):
########################################################################
# Converts a (grass)raster map layer theGrassRaster into an
#  ASCII text file theAsciiRaster
def rOutAscii(self,theGrassRaster,theAsciiRaster):
########################################################################
# constructs the input raster containing the coordinate of the WLF
# ignition point (theX,theY) the user selected
def wlfIgnitionPoint(self,theX,theY):
########################################################################
# removes old raster from previous simulations
def removeResultsFromPrevSimu(self):
########################################################################
# calculates live_moisture raster depending on amount of
# defoliation (read from the file theFilename) of the year theYear
def calcLiveMoisture(self,theYear,theFilename):
```

```
    ######################################################################
    # calulcate the the LiveMoisture value which is associated to the
    # defoliation value of the forest
    # cf mapping in dict self.liveMoistureAndDefol
    def calcLiveMoistureAccDef(self,theDefoliationValue):
    ######################################################################
    # Generates three raster map layers showing
    # 1) the base (perpendicular) rate of spread (ROS),
    # 2) the maximum (forward) ROS,
    # 3) the direction of the maximum ROS
    # prms:
    # input: no interactive: all input data is taken from self.grassRrosData
    # output: the 3 layers are stored in the Grass
    def rRos(self):
    ######################################################################
    # r.spread - Simulates elliptically anisotropic spread and generates a
    # raster map of the cumulative time of spread
    # input: no interactive. Input are
    # - raster maps containing the rates of spread (ROS),
    # - the ROS directions
    # - the spread origins
    def rSpread(self):
    ######################################################################
    # exports the result of the WLF spread from GRASS-DB to an ascii-raster
    # file on local disk
    # prms:
    # - the name of the grass raster is read from self.grassRspreadData
    # - the name of the ascii raster is read from self.wlfSpread
    # returns: name of ascii raster (holding the wlf spread)
    def wlfSpreadExport(self):
##############################################################################
if __name__ == "__main__":
    #
    ge = GisEvent()
    ge.processGisEvent(imCmd)
```

C.3.7 GuiEvent

The class `GuiEvent` is responsible for the communication of the IPODLAS system with the GUI legacy system. `GuiEvent` on the one hand catches events fired from the GUI legacy system and propagates them to the instance of the `IpodlasClient` deployed on the GUI subsystem and on the other hand forwards messages from the local `IpodlasClient` instance to the GUI legacy system. `GuiEvent` is defined in the Python module `guiEvent.py`.

```
##############################################################################
##############################################################################
# guiEvent.py
##############################################################################
# handles GUI events:
# - catches events from the GUI and sent them to the IpodlasClient
# - propagates IPODLAS events received by the local the IpodlasClient to the
# GUI
class GuiEvent:
    ######################################################################
    def __init__(self,theArgArray):
        #
        # id of this instance: e.g. GUI
        self.id = "GUI"
        #
        # current guiCommand: used to take current item from
        # config.guiRequestList: ["stopLbm","runLbm","runWlf"]
        self.guiRequest         #
        #
        self.lg = LoggingText()
    ######################################################################
```

```
    # parses startup args and sets member vars
    def parseStartUpArgs(self,theArgArray):
    ########################################################################
    # overwrites the lineReceived from base class
    def dataReceived(self, data):
    ########################################################################
    # processes a GUI event:e.g. a request to start visualizing LBMs
    def processGuiEvent(self,theArgArray):
    ########################################################################
    # processes the GUI event startLbm (i.e. start simulating and visu LBMs)
    # prm: the array containing the args handed over from GUI
    def processGuiStartLbmEvent(self,theArgArray):
    ########################################################################
    # processes the GUI event startWlf (i.e. start simulating and visu WLF)
    # prm: the array containing the args handed over from GUI
    def processGuiStartWlfEvent(self,theArgArray):
    ########################################################################
    # parses the arguments handed over from the GUI
    # for the GUI event startWlf (i.e. start simulating and visu WLF)
    # prm: the array containing the args handed over from GUI
    def parseGuiStartWlfArgs(self,theArgArray):
    ########################################################################
    # starts up the SSC, which invokes IK to generate a read request and
    # invoke the respective IC to read an info exchange file
    def startUpSsc(self):
##############################################################################
if __name__ == "__main__":
    ge = GuiEvent(sys.argv)
```

C.3.8 Log, Log_File, and LogFile

`Log` (and indirectly `Log_File`) is called by the other classes of the IPODLAS system to write their output timestamped into a common logfile. To get a timestamp showing milliseconds the class `Logfile` is invoked. These three classes are defined within the Python module logfile.py.

```
##############################################################################
##############################################################################
# logfile.py
##############################################################################
# writes output of IPODLAS classes timestamped to a logfile
# using receipt from ASPN
class Log:
     ########################################################################
    #
    class Log_File:
        ####################################################################
        def __init__(self, fPath = os.getcwd(), fName = "log.txt"):
            self.fpath      #  the class variable, file path
            self.fname      #  the class variable, file name
            self.log_file    #  the class variable, log file
            self.fpath
            self.fname
        ####################################################################
        def put(self,lineNo,text = ""):
        ####################################################################
        def get_id(self):
    ##############################################################################
    # class variables
    __log
    ##############################################################################
    def __init__(self, fPath = os.getcwd(), fName = "log.txt"):
    ##############################################################################
    def      get_id(self):
    ##############################################################################
    def      put(self, text = ""):
```

```
    ##############################################################################
    def      checkpt(self, text = ""):
    ##############################################################################
    def      write(self, text = ""):
##############################################################################
if __name__ == "__main__":
    pass

##############################################################################
##############################################################################
# logfile.py
##############################################################################
# class to get exact timestamp
class Logfile:
    ##########################################################################
    def __init__(self):
        # init logfile class: writes into logfile theLogfile
        self.logApp
        # which process invoked LoggingText
        self.invokingProcess
    ###############################################
    # returns a timestamp built from localtime (HH:MM:SS) with microsecs
    # from time()
    def getTimestamp(self):
    ##########################################################################
    # sends text theText to logging class self.logApp
    def write2Log(self,theText):
    ##########################################################################
    def setInvokingProcess(self,theInvokingProcess):
```

C.3.9 SubSysClient

The class `SubSysClient` is a helper class associated with the respective legacy system on the subsystems. When a legacy system have written output to a data file, the legacy system invokes an instance of `SubSysClient`, which establish a socket connection to responsible the `Ipodlaskernel` instance and triggers this instance to invoke the local `IpodlasClient` to read the data file. `SubSysClient` is derived from `IpodlasMessage`. `SSC_ClientFactory` is employed to instantiate `SubSysClient`. Both classes are defined in the Python module `subSysClient.py`.

```
##############################################################################
##############################################################################
# subSysClient.py
##############################################################################
# establish connection to IpodlasKernel over sockets and to subsystems via
file exchange
class SubSysClient(IpodlasMessage):
    ##########################################################################
    def __init__(self,theId):
    ##########################################################################
    # overwrites the lineReceived from base class
    def lineReceived(self, data):
    ##########################################################################
    # main control routine
    def clientControl(self, theData):
    ##########################################################################
    # identifies client: i.e. sends identifying receipt to ipodlasKernel
    def identify(self):
    ##########################################################################
    # requests from IK to invoke respective IC to read a local file
    def requestReadLocalFile(self):
##############################################################################
#
##############################################################################
# creates protocol and receives events relating to connection state
```

```
class SSC_ClientFactory(ReconnectingClientFactory):
    ########################################################################
    def __init__(self,theId):
        self.id = theId
    ########################################################################
    def startedConnecting(self, connector):
    ########################################################################

    def buildProtocol(self, addr):
    ########################################################################
    def clientConnectionLost(self, connector, reason):
    ########################################################################
    # if connection could not be established
    def clientConnectionFailed(self, connector, reason):
##########################################################################
# startup SSC_ClientFactory, in case of SSC_GUI this reactor is explictely
# stopped,in case of SSC_TSS not (since it seems to be  stopped somehow
# automatically??)
def startUpSubsystemClient(IK_ip,IK_port,ssc_id):
##############################################################################
if __name__ == "__main__":
     reactor.connectTCP("130.60.176.97",51423,SSC_ClientFactory(ID))
     reactor.run()
```

C.3.10 TssEvent

The class `TssEvent` is responsible for the communication of the IPODLAS system with the TSS legacy system. It receives requests from the `IpodlasClient` instance deployed on the TSS subsystem and invokes the TSS legacy system to handle the request and propagates the results back to the local `IpodlasClient` instance. `TssEvent` is defined in the Python module `tssEvent.py`.

```
##############################################################################
##############################################################################
# tssEvent.py
##############################################################################
# handles TSS events:
# - catches TSS requests sent from the IpodlasClient
# - propagates TSS results back to the IpodlasClient
class TssEvent:
    ########################################################################
    def __init__(self):
        self.id
        self.lg = LoggingText()# init logfile
    ########################################################################
    # read (information exchange) file of the associated subsystem,
    # e.g. infoEx_fromGUI.txt
    # return list of command elements:
    # e.g.cmdElementList ['simuLBM', 'time=1949-1957', 'site=1,20']
    def parseInfoEx_toTSS_File(self):
    ########################################################################
    # as workaround: to fake application of rRass, 2 output files of Ramses
    # are available:
    # config.ramsesResultFilenameList[0]: is calculated with default winter
    # egg mortality of 0.5728
    # config.ramsesResultFilenameList[1]: is calculated with winter egg
    # mortality of 0.65
    # in infoEx_toTSS: the user-selected winterEggMort is transmitted, which
    # is  e.g. winterEggMort=0.65
    # if winterEggMort < config.winterEggMortBorder, select
    # ramsesResultFilenameList[0], else ramsesResultFilenameList[1]
    def getRamsesResultFilename(self,theWinterEggMortString):
    ########################################################################
    # processes a TSS event: writes an IM to an infoExchange file and then
    # invoke an SSC to invoke IK via socket to request IC to read the
```

```
    # infoExchange file
    def processTssEvent(self):
################################################################################
if __name__ == "__main__":
    TssEvent()
```

C.3.11 VrEvent

The class `VrEvent` is responsible for the communication of the IPODLAS system with the VR legacy system. It receives requests from the `IpodlasClient` instance deployed on the VR subsystem and invokes the VR legacy system to handle the request and propagates the results back to the local `IpodlasClient` instance. `VR` is defined in the Python module `vrEvent.py`.

```
################################################################################
################################################################################
# vrEvent.py
################################################################################
# handles VR events:
# - catches TSS requests sent from the IpodlasClient
# - propagates TSS results back to the IpodlasClient
class VrEvent:
    ############################################################################
    def __init__(self):
        self.id
        self.lg = LoggingText()# init logfile
    ############################################################################
    # routine acting like a socket server: listening for incoming requests
    # and invoking processVrEvent() if VTP requests it
    def socketServer(self):
    ############################################################################
    # parses received data string theData
    # returns string returnValue : <handledHere|notHandledHere>
    def parseData(self,theData):
    ############################################################################
    # processes a VR event:e.g. writes a IM  in an infoWExchange file then
    # invoke an SSC to invoke IK via socket to request IC to read the
    # infoExchange file
    # input prm: theData : a string which contains x-,y-,z-coordinates
    def processVrEvent(self,theData):
    ############################################################################
    # encapsulate the starting up of the subsystem client so that it can be
    # done as thread
    def startUpSubsystemClient(self):
################################################################################
if __name__ == "__main__":
    #
    ve = VrEvent()
```

Bibliography

Albrecht, J., 1997. Universal GIS operations: a task-oriented systematization of data structure-independent GIS functionality leading towards a geographic modelling language. Geographic Information Research: Transatlantic Perspectives. Taylor and Francis, London, 577-591 pp.

Alexander, M., 1985. Estimating the length-to-breadth ratio of elliptical forest fire patterns, Proc. of the Eighth Conf. on Fire and Forest Meteorology. Soc. Am. For, Bethesda, MD, pp. 287–304.

Allgöwer, B., Fischlin, A. and Frei, U., 2003. Use case approach road map, Geographic Information Systems, Department of Geography, University of Zürich, http://bscw.geo.unizh.ch/bscw/bscw.cgi/d78284/use%20case%20approach%20roadmap.pdf (accessed October 10, 2005).

Allgöwer, B. et al., 2001. Knowledge based dynamic landscape analysis and simulation for alpine environments. Full Proposal for SNSF Project Nr.4048-064432, SNSF, Zürich, http://bscw.geo.unizh.ch/pub/bscw.cgi/d77727/nfp48_scient_final.pdf (accessed October 10, 2005).

Allgöwer, B., Harvey, S. and Rüegsegger, M., 1998. Fuel models for Switzerland: Description, spatial pattern, index for crowning and torching, 3rd International Conference on Forest Fire Research / 14th Conference on Fire and Forest Meteorology, Luso (Portugal), pp. 2605-2620.

Anderson, H., 1983. Predicting wind-driven wildland fire size and shape, USDA For. Serv. Res. Pap. INT-305.

Anderson, J.D., 1994. Computational fluid dynamics: the basics with applications. McGraw-Hill series in mechanical engineering. McGraw-Hill, New York, 547 p.

Anselin, L., 1989. What is special about spatial data? alternative perspectives on spatial data analysis, NCGIA, Santa Barabara.

Aspinall, R. and Pearson, D., 2000. Integrated geographical assessment of environmental condition in water catchments: Linking landscape ecology, environmental modelling and GIS. Journal of Environmental Management, 59(4): 299-319.

Babu, M., 2003. Implementing Internet GIS with Java based Client/Server Environment, Map Asia, http://www.gisdevelopment.net/technology/gis/pdf/ma03230.pdf (accessed April 25, 2006).

Bachmann, A., 1998. Coupling NUATMOS with the GIS ARC/INFO. Final Report for MINERVE 2, Department of Geography, University Zurich, Zurich, http://www.geo.unizh.ch/gis/research/edmg/fire/papers/minerve/prince.pdf (accessed October 10, 2005).

Baltensweiler, W. and Fischlin, A., 1988. The larch bud moth in the Alps. In: A.A. Berryman (Editor), Dynamics of forest insect populations: patterns, causes, implications. Population ecology. Plenum Publishing Corporation, New York a.o., pp. 331-351.

Baltensweiler, W. and Rubli, D., 1999. Dispersal: an important driving force of the cyclic population dynamics of the larch bud moth, Zeiraphera diniana Gn. For. Snow Landscape Res., 74(2): 3 - 153.

Barfield, L., 1993. The user interface: concepts & design. Addison-Welsey Pub. Co., Wokingham, England; Reading. Mass., 353 pp.

Batty, M., 1999. New Technology and GIS. Geographical Information Systems, Principles and Technical Issues. John Wiley & Sons, New York.

Batty, M. and Jiang, B., 1999. Multi-agent simulation: new approaches to exploring space-time dynamics within GIS, GISRUK 99 (Geographical Information Systems Research - UK), Southampton.

Bennett, D., 1997. A framework for the integration of geographical information systems and modelbase management. International Journal of Geographical Information Science, 11(4): 337-357.

Bergamin, J., 2003. IPODLAS - User interface study, Swiss Federal Institute of Technology: Zürich, Switzerland, http://bscw.geo.unizh.ch/bscw/bscw.cgi/d78858/User_interface_jb.pdf (accessed October 10, 2005).

Bergamin, J., 2004. Schnelle Datenübertragung für verteilte Simulation und Visualisierung. MSc Thesis, Swiss Federal Institute of Technology: Zürich, Switzerland, 43 pp.

Bergmann, A., Breunig, M., Cremers, A. and Shumilov, S., 2000a. A component based, extensible software platform supporting interoperability of GIS applications, Proceedings of the Umweltinformatik 2000 - Computer Science for Environmental protection. Metropolis-Verlag.

Bergmann, A., Breunig, M., Cremers, A. and Shumilov, S., 2000b. Towards an Interoperable Open GIS. In: M. Norrie, Laurini, R., Spaccapietra, S. (Editor), Intern. Workshop on emerging technologies for Geographical Information Systems for Geo-based Applications, Ascona, Switzerland, pp. 283-296, http://www.geo.informatik.uni-bonn.de/publications/2000/ascona2000.pdf (accessed in Dec 5, 2005).

Bernard, L., 2001. Integration von GIS und dynamischen Atmosphaerenmodellen auf Basis interoperabler objektorientierter Komponenten. PhD Thesis, Westfaelische Wilhelms-Universitaet Muenster, Muenster, 140 pp.

Bernard, L. and Krueger, T., 2000. Integration of GIS and spatio-temporal simulation models: interoperable components for different simulation strategies. Transaction in GIS, 4(3): 197 - 215.

Biegger, S., 2004. A visual system for the interactive study and experimental simulation of climate-induced 3D mountain glacier fluctuations. PhD Thesis, University of Zürich, 180 pp.

Bjork, R., 2004. Use Cases for Example ATM System. http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample (accessed January 09, 2006).

Bjornstad, O., Peltonen, M., Liebhold, A. and Baltensweiler, W., 2002. Waves of larch budmoth outbreaks in the European Alps. SCIENCE, 298(5595): 1020-1023.

Blazek, R. and Nardelli, L., 2004. The GRASS server, FOSS/GRASS Users Conference, Bangkok, Thailand, http://gisws.media.osaka-cu.ac.jp/grass04/viewpaper.php?id=30 (accessed May 16, 2006).

Blazek, R., Neteler, M. and Micarelli, R., 2002. The new GRASS 5.1 vector architecture, Open source GIS - GRASS users conference, Trento, Italy, http://www.ing.unitn.it/~grass/conferences/GRASS2002/proceedings/proceedings/pdfs/Blazek_Radim.pdf (accessed January 16, 2006).

Boehm, B., 1988. A Spiral Model of Software Development and Enhancement. Computer, 21(5): 61-72.

Brandmeyer, J.E. and Karimi, H.A., 2000. Coupling methodologies for environmental models. Environmental Modelling & Software, 15(5): 479-488.

Brimicombe, A., 2003. GIS, environmental modelling and engineering. Taylor & Francis, London; New York, 312 pp.

Brooks, F.P., 1987. No Silver Bullet - Essence and Accidents of Software Engineering (Reprinted from Information-Processing 86, 1986. Computer, 20(4): 10-19.

Buehler, K. and McKee, L., 1998. The OpenGIS Guide. Open GIS Consortium Technical Commitee, Wayland.

Burrough, P., 1988. Linking spatial process models and GIS: a marriage of convenience of a blossoming partnership? GIS/LIS. ASPRS/ACSM, Falls Church, VA., pp. 598 - 607.

Cagle, K. et al., 2001. Professional XSL. Wrox Press Ltd., Birmingham, UK, 800 pp.

Camara, A.S. et al., 1998. Virtual Environments and Water Quality Management. Journal of Infrastructure Systems, 4(1): 28-36, http://link.aip.org/link/?QIS/4/28/1.

Cellier, F., 1991. Continuous system modeling. Tucson, Arizona 85721, http://www.ece.arizona.edu/~cellier/springer.html (accessed November 28, 2005).

Chang, Y. and Park, H., 2006. XML Web Service-based development model for Internet GIS applications. International Journal of Geographical Information Science, 20(4): 371, http://journalsonline.tandf.co.uk/openurl.asp?genre=article&id=doi:10.1080/13658881060060785 7 (accessed May 19, 2006).

Chen, J. and Jiang, J., 2000. An Event-Based Approach to Spatio-Temporal Data Modeling in Land Subdivision Systems. GeoInformatica, 4(4): 387-402, http://www.springerlink.com/index/10.1023/A:1026565929263.

Chrismann, N., 1998. Beyond the Snapshot: Changing the Approach to Change, Error, and Process. In: M. Egenhofer and R. Golledge (Editors), Spatial and Temporal Reasoning in Geographic Information Systems. Oxford University Press, Oxford, UK, pp. 85 - 93.

Coleman, D., 1999. Geographical Information Systems in networked environments. In: P. Longley, Goodchild, M., Maguire,D. and, Rhind, D. (Editor), Geographical Information Systems. John Wiley & Sons, New York, pp. 317 - 329.

Couclelis, H., 1999. Space, time, geography. In: P. Longley, Goodchild, M., Maguire,D. and, Rhind, D. (Editor), Geographical Information Systems. John Wiley & Sons, New York, pp. 29 -38.

Crosswell, P., 2000. The role of standards in support of GDI. Geospatial Data Infrastructures - Concepts, Cases and Good Practice. Oxford University Press, 57 - 83 pp.

Curbera, F., Nagy, W. and Weerawarana, S., 2001. Web Services: Why and How. http://researchweb.watson.ibm.com/people/b/bth/OOWS2001/nagy.pdf (accessed January 09, 2006).

D'Ambrogio, A. and Gianni, D., 2004. Using CORBA to enhance HLA interoperability in distributed and Web-based simulation. Computer and Information Sciences - Iscis 2004, Proceedings, 3280: 696-705.

De Vasconcelos, M.J.P., Goncalves, A., Catry, F.X., Paul, J.U. and Barros, F., 2002. A working prototype of a dynamic geographical information system. International Journal of Geographical Information Science, 16(1): 69-91.

Downey, A.B., 2001. How to think like a computer scientist: learning with python. LuluPress, Durham, NC, 288 pp.

Duchaineau, M. et al., 1997. ROAMing Terrain: Real-time Optimally Adapting Meshes, IEEE Visualization 97, Phoenix, USA, pp. 81–88.

Eckel, B., 2001. Thinking in Python. http://www.mindview.net/Books/TIPython (accessed January 11, 2006).

Eisenberg, J.D., 2002. SVG essentials. O'Reilly, Sebastopol, CA, 335 pp.

Engelen, G., Geertman, S., Smits, P. and Wessels, C., 1999. Dynamic GIS and Strategic Physical Planning: A Practical Application. Geographical information and planning. Springer, Berlin, 87 - 111 pp.

Englander, R., 2002. Java and SOAP. O'Reilly, Sebastopol, CA, 258 pp.
ESRI, 1993. ARC Macro Language Developing ARC/INFO menus and macro with AML. Environmental Systems Research Institute, Inc., Redlands, CA.
ESRI, 2005. Raster data in ArcSDE 9.1. http://www.esri.com/library/whitepapers/pdfs/arcsde91-raster.pdf (acessed May 09, 2006).
Fedra, K., 1993. Gis and environmental modeling. In: M. Goodchild, Parks, B., Steyaert, L. (Editor), Environmental modeling with GIS. Oxford University Press, New York, pp. 35-50.
Fedra, K., 1996. Distributed models and embedded GIS: integration strategies and case studies. In: M. Goodchild, Steyaert, L., Parks, B. (Editor), GIS and environmental modeling: progress and research issues. GIS World Books, Fort Collins, CO, pp. 413 - 418.
Feiler, J., 2000. Application server: Powering the Web-Based Enterprise. Academic Press, San Diego, CA, 302 pp.
Fettig, A., 2005. Twisted Network Programming Essentials. O'Reilly Media, Inc., 236 pp.
Finney, M., 1998. FARSITE: Fire Area Simulator-Model Development and Evaluation, USDA United States Department of Agriculture, Forest Service.
Finney, M., 2004. FARSITE, fire area simulator--model development and evaluation. Research paper RMRS; RP-4. U.S. Dept. of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, UT (324 25th St., Ogden 84401), 47 pp.
Fischer, W., 1982. Photo guide for appraising downed woody fuels in Montana forests: Grand Fir-Lach-Douglas-Fir, Western Hemlock, Western Hemlock Western Redcedar, and Western Redcedar cover types, USDA Forest Service, Intermountain Forest and Range Experiment Station, Ogden, UT.
Fischlin, A., 1982. Analyse eines Wald-Insekten-Systems: Der subalpine Lärchen-Arvenwald und der graue Lärchenwickler Zeiraphera diniana GN. (Lep., Tortricidae). Diss. ETH Thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 294 pp.
Fischlin, A., 1983. Modelling of Alpine valleys, defoliated forests, and larch bud moth cycles: the role of migration. In: R.H. Lamberson (Editor), Mathematical models of renewable resources. Humboldt State University, Mathematical Modelling Group, University of Victoria, Victoria, B.C., Canada, pp. 102-104.
Fischlin, A., 1991. Interactive modeling and simulation of environmental systems on workstations. In: D. Moeller (Editor), Analysis of dynamic systems in medicine, biology, and ecology,Informatik-Fachberichte, pp. 131-145.
Fischlin, A. and Baltensweiler, W., 1979. Systems analysis of the larch bud moth system. Part I: the larch-larch bud moth relationship. Mitteilungen der Schweizerischen Entomologischen Gesellschaft, 52: 273-289.
Fischlin, A. et al., 2002a. Use Case TreeGrowth for IPODLAS Prototype 0.0. Workshop Nov 2002, internal report, http://bscw.geo.unizh.ch/pub/bscw.cgi/d77716/IPODLAS_UC_TreeGrowth_v0.6.pdf (accessed October 10, 2005).
Fischlin, A., Roth, O., Gyalistras, D. and Ulrich, M., 2002b. ModelWorks., Zürich, http://www.sysecol.ethz.ch/SimSoftware/RAMSES/Documents/On_ModelWorks-READ_ME.pdf (accessed in January 02, 2006).
Flanagan, D., 2005. Java in a nutshell, Fifth Edition. O'Reilly Media, Inc., Sebastopol, CA 95472, 1252 pp.
Frank, A., 1998. Different types of "Times" in GIS. Spatial and temporal reasoning in Geographic Information Systems. Oxford University Press, New York, NY, 40 - 62 pp.

Franklin, S. and Graesser, A., 1997. Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. Intelligent Agents III: Agent Theories, Architectures, and Languages. Springer Verlag, Berlin, 21-35 pp.

Frehner, M., Brändli, M. and Schenker, J., 2004. The Virtual Database – A Tool for Integrated Data Process-ing in a Distributed Environment, 18th International Conference Informatics for Environmental Protection, Geneva (Switzerland), http://www.wsl.ch/staff/marcel.frehner/publications/theVirtualDatabase_frehner_braendli_schenker_enviroinfo2004.pdf (accessed May 19, 2006).

Fujimoto, R., 1999. Parallel and distributed simulation systems. Wiley series on parallel and distributed computing. John Wiley & Sons, New York, 320 pp.

Gerhardt, M. and Schuster, H., 1995. Das digitale Universum: zellulaere Automaten als Modelle der Natur. Vieweg Verlag, Braunschweig, 320 pp.

Ghezzi, C., Jazayeri, M. and Mandrioli, D., 2003. Fundamentals of software engineering. Prentice Hall, Upper Saddle River, N.J., 604 pp.

Giorgetta, F., 2002. Integration von Raum und Zeit in ein Landschaftsanalyse und Simulationssystem: Systemtheoretische Grundlagen und Evaluation gängiger Systemsimulationsprogramme. Semesterarbeit Thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 54 pp.

Goddard, S., Zhang, S., Waltman, W. and Lytle, D., 2002. A Software Architecture for Distributed Geospatial Decision Support Systems, National conference for digital government research, pp. 45 -52, http://cse.unl.edu/~goddard/Papers/Conference/dgo2002.pdf (accessed June 14, 2006).

Goerzen, J., 2004. Foundations of Python network programming. The expert's voice in open source. Apress: Distributed to the Book trade in the United States by Springer-Verlag, Berkeley, CA,New York, 512 pp.

Gold, C., Chau, M., Dzieszko, M. and Goralsk, R., 2004a. The Marine GIS - Dynamic GIS in action. In: P. Fisher (Editor), XXth ISPRS Congress, Instanbul, pp. 97-108, http://www.isprs.org/istanbul2004/comm2/papers/215.pdf (accessed February 6, 2006).

Gold, C., Chau, M., Dzieszko, M. and Goralski, R., 2004b. 3D Geographic Visualization: The Marine GIS. In: P. Fisher (Editor), Developments in Spatial Data Handling - 11th International Symposium on Spatial Data Handling. Springer, pp. 97-108.

Goodchild, M., Steyaert, L., Parks, B., 1996. GIS and environmental modeling: progress and research issues. GIS World Books, Fort Collins, CO, 486 pp.

Gronmo, R., Berre, A., Solheim, I., Hoff, H. and Lantz, K., 2000. DISGIS: An Interoperability Framework for GIS - Using the ISO/TC 211Model-based Approach. http://heim.ifi.uio.no/~roygr/GSDI-2000.pdf (accessed December 1, 2005).

Haklay, M., 2001. Virtual Reality and GIS. Applications Trends and Directions. Taylor and Francis, London.

Harold, R. and Means, W., 2004. XML in a nutshell. O'Reilly, Cambridge, 718 pp.

Harvey, S., Ruegsegger, M. and Allgöwer, B., 1997. Fuel models for Switzerland (Swiss National Park), MINERVE, Saint-Paul les Durence Cedex, France, http://www.geo.unizh.ch/gis/research/edmg/fire/papers/minerve/fuel.pdf (accessed March 20, 2006).

Hirtz, P., Hoffmann, H.., Nüesch, D., 1999. Interactive 3D landscape visualization: improved realism through use of remote sensing data and geoinformation. Proc. Computer Graphics International, pp. 101 – 108.

Hodgin, R., Ciolek, J., Buckley, D. and Bouwman, D., 1997. Integrating atmospheric dispersion modelling with ARC/INFO: a case study of the regional atmospheric response center, ESRI International User Conference, Denver, CO.

Hoheisel, A., 2002. SWIM meets XML - The Man Model Measurement (M3) project. http://mmm.first.fhg.de/papers/hoheisel_m3_iemss2002.pdf (accessed October 10, 2005).

Huang, B., Jiang, B. and Li, H., 2001. An integration of GIS, virtual reality and the Internet for visualization, analysis and exploration of spatial data. International Journal of Geographical Information Science, 15(5): 439-456.

I.Sommerville, 2001. Software engineering. Addison-Wesley, 711 pp.

IEEE, 1990. IEEE Standard Computer Dictionary: A compilation of IEEE Standard Computer Glossaries, Institute of Electrical and Electronics Engineers, Inc., New York.

Isenegger, D., Price, B. and Wu, Y., 2004. Crossimplementation of wildland fire and larch bud moth models in GIS and Ramses, Geographic Information Systems, Department of Geography, University of Zürich, http://bscw.geo.unizh.ch/pub/bscw.cgi/d77674/x_implementation.pdf (accessed October 10, 2005).

Isenegger, D. et al., 2005. IPODLAS - A software architecture for coupling temporal simulation systems, VR, and GIS. ISPRS Journal of Photogrammetry and Remote Sensing, 60(1): 34-47, http://www.sciencedirect.com/science/article/B6VF4-4HK04HB-1/2/1857645a2462ced36547c3902a1c8528.

Jacobson, I., Booch, G. and Rumbaugh, J., 1999. The Unified Software Development Process. Addison-Wesley, Reading, Massachusetts, 463 pp.

Jecklin, R. and Schöb, T., 1993. Waldbrandbekämpfung im Gebirge. Verlag Bündner Wald, Chur.

Johnson, A., 1996. Spatiotemporal Hierarchies in Ecological Theory and Modelling. GIS and Environmental Modeling: Progess and Research Issues. GIS World Books, Fort Collins, CO 80525.

Jones, C., 1997. Geographical Information Systems and Computer Cartography. Addison Wesley Longman, Harlow, 319 pp.

Jones, C.A. and Drake, F.L., 2002. Python and XML. O'Reilly, Sebastopol, Calif., 360 pp.

Karimi, H.A. and Houston, B.H., 1996. Evaluating strategies for integrating environmental models with GIS: Current trends and future needs. 20(6): 413, http://www.sciencedirect.com/science/article/B6V9K-3VV038X-J/2/311f2cd45d71c1aaf01551b6b9d27ba7.

Kim, D. and Kim, M., 2002. Web GIS service component based on open environment. Geoscience and Remote Sensing Symposium, 6: 3346 - 3348, http://ieeexplore.ieee.org/iel5/7969/22041/01027178.pdf?isnumber=&arnumber=1027178 (accessed May 17, 2006).

Kiwon, L., Sun Hee, M. and Byung-Doo, K., 2003. GML-based representation architecture for digital geo-science GIS layers: a case study using Korea digital geologic map sets, pp. 3568.

Knuth, D.E., 1997. The art of computer programming. Addison-Wesley, Reading, Mass.

Kotonya, G. and Sommerville, I., 1998. Requirements Engineering, Processes and Techniques. J.Wiley & Sons, Chichester, 288 pp.

Kraak, M., 2003. Geovisualization illustrated. ISPRS Journal of Photogrammetry and Remote Sensing, 57(5-6): 390-399.

Kraak, M., Smets, G. and Sidjanin, P., 1999. Virtual Reality, The New 3-D Interface for Geographical Information Systems. Spatial Multimedia and Virtual Reality. Taylor and Francis, London, 131-136 pp.

Kuhn, M., 1989. Föhnstudien. Wissenschaftliche Buchgesellschaft, Darmstadt.

Kwan, M. and Lee, J., 2003. Geovisualization of Human Activity Pattern Using 3D GIS: A Time-Geographic Approach. Spatially Integrated Social Science: Example in Best Practice. Oxford University Press, Oxford.

Lake, R., 2001. GML 2.0 - Enabling the Geo-spatial Web. http://www.geojava.net/company/galdos/articles/GML3.htm (accessed October 10, 2005).

Lake, R., Burggraf, D., Trinic, M. and Rae, L., 2004. GML Geography Mark-Up Language. J.Wiley & Sons, Southern Gate, Chichester, West Sussex, 388 pp.

Lamorlette, A. and Foster, N., 2002. Structural modeling of flames for a production environment. Acm Transactions on Graphics, 21(3): 729-735.

Landis, J., 2001. CUF, CUF II, and CURBA: A family of spatially explicit urban growth and land-use policy simulation models. Planning support systems: integrating geographic informations systems, models, and visualization tools. ESRI, Redlands, CA., 157 - 200 pp.

Langran, G., 1992. Time in Geographic Information Systems. Technical Issues in Geographical Information Systems. Taylor & Francis, Bristol, 189 pp.

Leclercq, E., Benslimane, D. and Yetongnon, K., 1996. A Distributed Object Architecture for Interoperable GIS, Ninth International Conference on Parallel and Distributed Computing Systems.

Leclercq, E., Benslimane, D. and Yetongnon, K., 1997. Interoperability of GIS: a Distributed Object Architecture based on a canonical and Functional Model, 16.International Association of Science and Technology for Development (IASTED'96). The International Association of Science and Technology for Development (IASTED), Innsbruck, Austria, pp. 40-43.

Lefkowitz, G. and Shtull-Trauring, I., 2003. Network Programming for the Rest of Us, USENIX Annual Technical Conference, FREENIX Track, San Antonio, Texas, pp. 77 - 90, https://www.usenix.org/events/usenix03/tech/freenix03/full_papers/lefkowitz/lefkowitz_html/ (accessed January 20, 2006).

Lindstrom, P. et al., 1997. An integrated Global GIS and Visual Simulation System, Atlanta., ftp://ftp.gvu.gatech.edu/pub/gvu/tech-reports/1997/97-07.pdf (accessed October 10, 2005).

Longley, P., Goodchild, M., Maguire, D. and Hind, R., 1999a. Spatial databases. Geographic Information Systems, 1. John Wiley & Sons, New York.

Longley, P., Goodchild, M., Maguire, D. and Rhind, D., 1999b. Introduction. In: P. Longley, M. Goodchild, D. Maguire and D. Rhind (Editors), Geographic Information Systems. John Wiley & Sons, New York.

Löwgren, J., 1993. Human-computer interaction. What every system developer should know. Studentlitteratur, Lund, Sweden.

MacEachren, A. and Monmonier, M., 1992. Geographic Visualization: Introduction. Cartography and Geographic Information Systems, 19(4): 197 - 200.

MacEachren, A., Wachowicz, M., Edsall, R., Haug, D. and Masters, R., 1999. Constructing knowledge from multivariate spatiotemporal data: integrating geographical visualization with knowledge discovery in database methods. International Journal of geographical information system, 13(4): 311-334.

Manninger, M., Göschka, K., Schwaiger, C. and Dietrich, D., 2001. Electronic Commerce - die Technik: Technologie, Design und Implementierung. Hüthig, Heidelberg.

Mardia, K.V. and Jupp, P.E., 2000. Directional statistics. Wiley series in probability and statistics. J. Wiley, Chichester; New York, 429 pp.

Martelli, A., 2003. Python in a nutshell. O'Reilly Media, Inc., Sebastopol, CA 95472, 654 pp.

McCormick, B.H., Defanti, T.A. and Brown, M.D., 1987. Visualization in Scientific Computing. IEEE Computer Graphics and Applications, 7(10): 69-69.

Meyer, A., Neyret, F. and Poulin, P., 2001. Interactive Rendering of Trees with Shading and Shadows, London, England, http://artis.imag.fr/Publications/2001/MNP01/MNP01.pdf (accessed October 10, 2005).

Mladenoff, D., Host, G., Boeder, J. and Crow, T., 1996. LANDIS: a spatial model of forest landscape disturbance, succession, and management. GIS and environmental modeling: progress and research issues. GIS World Books, Fort Collins, CO, 175 -180 pp.

Möller, T. and Haines, E., 1999. Real-time rendering. A K Peters, Natick, Mass., 482 pp.

Naur, P. and Backus, J.W., 1962. Bericht über die algorithmische Sprache ALGOL, 60. Elektronisches Rechnen und Regeln, Bd. 1. Akademie-Verlag, Berlin, 49 pp.

Neteler, M. and Mitasova, H., 2002. OPEN SOURCE GIS: a GRASS GIS approach. Kluwer Academic Publishers, Boston, 419 pp.

Neumann, K. and Eckstein, S., 2002. Geography Markup Language (GML) - Eine Einführung aus Informatiksicht. http://www.ifis.cs.tu-bs.de/html_d/home/neumann/vortraege.html (accessed January 13, 2006).

Neves, J., Goncalves, P., Muchaxo, J. and Silva, J., 1999. A virtual GIS room: interfacing spatial information in virtual environments. Spatial Multimedia and Virtual Reality. Taylor and Francis, London, 149-157 pp.

Nievergelt, J. and Weydert, J., 1980. Sites, Modes, and Trails: Telling the user of an interactive system where he is, what he can do, and how to get to places. Methodology of Interaction. North-Holland Publishing Company, 327 - 338 pp.

Nyerges, T., 1993. Understanding the scope of GIS: its relationship to environmental modeling. Environmental modeling with GIS. Oxford University Press, New York, 75 - 107 pp.

O'Sullivan, D. and Unwin, D., 2002. Geographic information analysis. Wiley, Hoboken, N.J., 436 pp.

OMG, 1999. CORBA components and component model, document orbos/99-02-05. http://www.omg.org/ (accessed October 10, 2005).

Onsrud, H., 1989. Understanding the uses and assessing the value of geographic information, GIS/LIS. ASPRS/ACSM, Falls Church, VA., pp. 404 - 411.

Pajarola, R. and Widmayer, P., 2001. Virtual geoexploration: Concepts and design choices. International Journal of Computational Geometry & Applications, 11(1): 1-14.

Pang, M.Y.C. and Shi, W.Z., 2002. Development of a process-based model for dynamic interaction in spatio-temporal GIS. Geoinformatica, 6(4): 323-344.

Percivall, G. , 2002. The OpenGIS Abstract Specification, Topic 12: OpenGIS Service Architecture, Version 4.3.OpenGIS Consortium, Wayland.

Peuquet, D., 1999. Time in GIS and geographical databases. In: P. Longley, Goodchild, M., Maguire,D. and, Rhind, D. (Editor), Geographical Information Systems. John Wiley & Sons, New York, pp. 91 -103.

Peuquet, D. and Duan, N., 1995. An Event-Based Spatiotemporal Data Model (ESTDM) for Temporal Analysis of Geographical Data. International Journal of Geographical Information Systems, 9(1): 7-24.

Peuquet, D.J., 2000. Time in GIS and geographical databases. Geographical Information Systems, Principles and Technical Issues. John Wiley & Sons, New York.

Peuquet, D.J., 2001. Making space for time: Issues in space-time data representation. Geoinformatica, 5, 11-32 pp.

Pilgrim, M., 2004. Dive into Python. The expert's voice in open source. Apress: Distributed to the Book trade in the United States by Springer-Verlag, Berkeley, CA., New York, 413 pp.

Praehofer, H., 1992. System theoretic foundation for combined discrete continuous system simulation, Johannes Kepler University, Linz.

Preston, M., Clayton, P. and Wells, G., 2003. Dynamic run-time application development using CORBA objects and XML in the field of distributed GIS. International Journal of Geographical Information Science, 17(4): 321-341.

Price, B., 2005. Spatio-temporal modelling and analysis of larch bud moth population dynamics in the European Alps. Ph. D. Thesis, Eidgenössische Technische Hochschule, Zürich, Switzerland, 194 pp.

Price, B., Allgöwer, B. and Fischlin, A., in press. Synchrony and travelling waves of Larch Bud Moth? Time series analysis with changing scale. Ecological Modelling.

Price, B., Isenegger, D., Allgöwer, B. and Fischlin, A., submitted. Spatio-temporal modelling of Larch bud moth in the European Alps: The importance of data resolution. Landscape Ecology.

Price, B., Isenegger, D. and Wu, Y., 2003. Models for IPODLAS case studies, http://bscw.geo.unizh.ch/pub/bscw.cgi/d77699/Models_IPODLAS_Case_studies.pdf (accessed October 10, 2005).

Price, B., Isenegger, D. and Wu, Y., 2005. Use Case Larch Budmoth Expert 2, http://bscw.geo.unizh.ch/pub/bscw.cgi/d77703/use_case_LE2_v1_2.pdf (accessed October 10, 2005).

Quattrochi and Goodchild, M., 1997. Scale, Multiscaling, Remote Sensing, and GIS. In: Quattrochi and M. Goodchild (Editors), Scale in remote sensing and GIS. Lewis Publishers, Boca Raton, Fla., pp. 1 - 12.

Raper, J. and Livingstone, D., 1995. Development of a geomorphological spatial model using object-oriented design. International Journal of Geographical Information Science, 9(4): 359-383.

Rembold, U. and Levi, P., 1999. Einführung in die Informatik. Hansen, München.

Rhyne, T.M., 1997. Going virtual with geographic information and scientific visualization. Computers & Geosciences, 23(4): 489-491.

Riedemann, C. and Timm, C., 2003. Services for data integration. Data Science Journal, 2: 75 - 83.

Robertson, P., 1990. A methodology for scientific data visualisation: choosing representations based on a natural scene paradigm, IEEE Conference on Visualisation '90, pp. 56-67.

Robertson, P., 1991. A Methodology for Choosing Data Representations. IEEE Computer Graphics and Applications, 11(3): 56-67.

Ross, D., Krautschneider, M., Smith, I. and Lorimer, G., 1988. Diagnostic wind field modeling: Development and validation, Centre for Applied Mathematical Modelling, Chisholm Institute of Technology.

Rothermel, R., 1972. A mathematical model for predicting fire spread in wildland fuels, USDA Forest Service, Intermountain Forest and Range Experiment Station.

Rubin, T., 1988. User interface design for computer systems. E. Horwood; Halsted Press, Chichester, West Sussex, New York, 195 pp.

Rushing, S., 2005. Asynchronous socket programming. http://squirl.nightmare.com/medusa/async_sockets.html (accessed January 16, 2006).

Sauer, H., 1994. Relationale Datenbanken: Theorie und Praxis. Addison-Wesley, Bonn, 291 pp.

Savary, L. and Zeitouni, K., 2003. Spatial data warehouse - A prototype. Electronic Government, Proceedings, 2739, 335-340 pp.

Schöning, R., 1996. Modellierung des potentiellen Waldbrandverhaltens mit einem GIS. MSc Thesis, University of Zurich, Zuerich, Switzerland, 140 pp.

Schulze, T., Wytzisk, A., Simonis, I. and Raape, U., 2002. Distributed spatio-temporal modeling and simulation. In: E. Yuecesan, Chen, C., Snowdon, J., Charnes, J. (Editor), Winter simulation conference, San Diego, CA, pp. 695 - 703.

Shapiro, M., 1991. r.mapcalc. http://grass.itc.it/grass60/manuals/html60_user/r.mapcalc.html (accessed February 21, 2006).

Shi, W. and Pang, M., 2000. Development of Voronoi-based cellular automata - an integrated dynamic model for Geographical Information Systems. International Journal of Geographical Information Science, 14(5): 455-474.

Shi, X., 2004. Client-Server Interaction in GIS Applications Through Web Services, ESRI International User Conference, West Virgina University, http://gis.esri.com/library/userconf/proc04/docs/pap1680.pdf (accessed May 19, 2006).

Shrestha, B., 2004. XML Database Technology and its use for GML. MSc Thesis, ITC, Enschede, 96 pp.

Simonis, I. and Wytzisk, A., 2003. Interoperability of Simulation and Geoinformation Standards, Spring Simulation Interoperability Workshop, Orlando, http://ifgi.uni-muenster.de/~simonis/publications.htm. (accessed 23.Nov.2005).

Stevens, W., 1990. UNIX Network Programming. Prentice Hall P T R, Englewood Cliffs, NJ, 772 pp.

Steyaert, T., 1993. A perspective on the state of environmental simulation modeling. In: M. Goodchild, Parks, B., Steyaert, L. (Editor), Environmental modeling with GIS. Oxford University Press, New York, pp. 16-30.

Stommel, H., 1963. Varieties of oceanographic experiments. Science, 139: 572-576.

Sui, A. and Maggio, R., 1999. Integrating GIS with hydrological modeling: practices, problems and prospects. Computers, Environment and Urban Systems, 23: 33 - 51.

Szyperski, C., 1998. Emerging component software technologies - a strategic comparison. Software - Concepts & Tools, 19(1): 2, http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s003780050002.

Szyperski, C., Gruntz, D. and Murer, S., 2003. Component software: beyond object-oriented programming. Addison-Wesley, London; Boston, MA.

Thöny, J., Fischlin, A. and Gyalistras, D., 1995. Introducing RASS - The RAMSES Simulation Server. http://www.sysecol.ethz.ch/Articles_Reports/Th23.pdf (accessed February 17, 2006).

Tidwell, D., 2001. XSLT. O'Reilly, Cambridge [Mass.]. 460 pp.

Tobler, W., 1970. A computer movie: simulation of population change in the Detroit Region. Economic Geography, 46: 234-240.

Tomlin, C.D., 1990. Geographic information systems and cartographic modeling. Prentice Hall, Englewood Cliffs, N.J., 249 pp.

Ungerer, M.J. and Goodchild, M.F., 2002. Integrating spatial data analysis and GIS: a new implementation using the Component Object Model (COM). International Journal of Geographical Information Science, 16(1): 41-53.

UserLand Software, I., 2003. XML-RPC Home Page. http://www.xmlrpc.com/ (accessed October 10, 2005).

Van Dam, A., Forsberg, A., Laidlaw, D., LaViola, J. and Simpson, R.M., 2000. Immersive VR for scientific visualization: a progress report. Computer Graphics and Applications, IEEE, 20(6): 26 - 52, http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=888006 (accessed November 22, 2005).

Vckovski, A., 1998. Interoperable and distributed processing in GIS, Taylor & Francis, London; Bristol, PA, 230 pp.

Verbree, E., Verzijl, L. and Kraak, M., 1998. Use of Virtual Reality and 3D-GIS within the Planning Process Concerning the Infrastructure. University of Otago, New Zealand, http://divcom.otago.ac.nz/sirc/webpages/Conferences/SIRC98/98Abstracts/98Verbee/Verbree.pdf (accessed January 2, 2006).

Voisard, A. and Schweppe, H., 1998. Abstraction and decomposition in interoperable GIS. International Journal of Geographical Information Science, 12(4): 315-333.

Wang, F., 2000. A Distributed Geographic Information System on the Common Object Request Broker Architecture. GeoInformatica, 4(1): 89 - 115.

Wang, X., 2004. Integrating GIS, simulation models, and visualization in traffic impact analysis. Computers, Environment and Urban Systems, 29: 471 - 496.

Westervelt, J. and Shapiro, M., 2000. Combining Scientific Models into Management Models, 4th International Conference on Integrating GIS and Environmental Modeling (GIS/EM4), Banff, Alberta, Canada.

Wiederhold, G., 1992. Mediators in the architecture of future information systems. IEEE Computer, 25(3): 38 - 49.

Wiederhold, G., 1995. The conceptual basis for mediation services. IEEE Intelligent Systems and Their Applications, 12(5): 38 - 47.

Williams, N., 1999. Four-dimensional virtual reality GIS (4D VRGIS): research guidelines. In: B. Gittings (Editor), Innovations in GIS 6. Taylor and Francis, London.

Windle, D. and Abreo, R., 2003. Software requirements using the unified process: a practical approach. Pearson Education, Inc., published as Prentice Hall PTR, Upper Saddle River, New Jersey.

Wirz, D., 2001. Technologies for semistructured Geodata. http://www.geo.unizh.ch/publications/wirz/brownbag_xml_c.pdf (accessed January 11, 2006).

Wittmann, J., 2000. Simulationsmodell und Geographisches Informationssytem: Koppelungsalternativen am praktischen Beispiel. In: A. Cremers and K. Greve (Editors), Umweltinformatik 2000, 12. Internationales Symposium. Metropolis-Verlag, Bonn, pp. 45-48.

Wood, J., 2002. Java programming for spatial sciences. Taylor & Francis, London; New York, 320 pp.

Wood, J.D., Fisher, P.F., Dykes, J.A., Unwin, D.J. and Stynes, K., 1999. The use of the landscape metaphor in understanding population data. Environment and Planning B-Planning & Design, 26(2): 281-295.

Worboys, M., 1999. Relational databases and beyond, in Geographic Information Systems. Principles and technical issues, Volume 1. John Wiley & Sons, New York.

Wu, Y., Biegger, S., Allgöwer, B. and Nuesch, D., submitted. Efficient cloud rendering, Eurographics'2006.

Wu, Y. et al., accepted. Real-time 4D visualisation of migratory insect dynamics within an integrated spatio-temporal system. Ecological Informatics.

Wytzisk, A., 2003. Interoperable Geoinformations- und Simulationsdienste auf Basis internationaler Standards. PhD Thesis, Westfaelische Wilhelms-Universitaet Muenster, 134 pp.

Xu, J., 1994. Simulating the spread of wildfires using a geographic information system and remote sensing. Ph. D. Thesis, Rutgers University, New Brunswick, New Jersey.

Xuan, Z., Richard, G.H. and Richard, J.A., 1998. A Knowledge-Based Systems Approach to Design of Spatial Decision Support Systems for Environmental Management. Environmental Management, 22(1): 35, http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s002679900082.

Zaslavsky, I., Marciano, R., Gupta, A. and Baru, C., 2000. XML-based Spatial Data Mediation Infrastructure for Global Interoperability, 4th Global Spatial Data Infrastructure Conference, Cape Town, South Africa.

Zeigler, B.P., 1976. Theory of modeling and simulation. Wiley, New York, 435 pp.

Zeigler, B.P., 1990. Object-oriented simulation with hierarchical, modular models: intelligent agents and endomorphic systems. Academic Press, Boston, 395 pp.

Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
ISPRS Journal of Photogrammetry & Remote Sensing 60 (2005) 34–47
PHOTOGRAMMETRY & REMOTE SENSING
www.elsevier.com/locate/isprsjprs

IPODLAS—A software architecture for coupling temporal simulation systems, VR, and GIS

Daniel Isenegger [a,*], Bronwyn Price [b], Yi Wu [c], Andreas Fischlin [b], Urs Frei [c], Robert Weibel [a], Britta Allgöwer [a]

[a] *GIS Division, Department of Geography, University of Zurich, Winterthurerstr. 190, 8057 CH—Zurich, Switzerland*
[b] *Institute of Terrestrial Ecology (Terrestrial Systems Ecology), ETH Zurich (ITÖ), Switzerland*
[c] *Remote Sensing Laboratories, Department of Geography, University of Zurich, Switzerland*

Received 6 October 2005; received in revised form 10 October 2005; accepted 11 October 2005
Available online 15 November 2005

Abstract

Environmental processes often vary in space and time and act over several scales. Current software applications dealing with aspects of these processes emphasize properties specific to their domain and tend to neglect other issues. For example, GIS prefers a static view and generally lacks the representation of dynamics, temporal simulation systems emphasize the temporal component but ignore space to a great extent, and virtual reality tends to "forget" the underlying data and models. In order to remedy this situation we present an approach that aims to bring together the three domains; temporal simulation systems, GIS, and virtual reality, and to foster the integration of particular functionalities. This paper concentrates on concepts and requirements for the development of a suitable software architecture using case studies and use cases seen from a GIS-based perspective.

Keywords: GIS; design; integration; simulation; visualization; interoperability; software

1. Introduction and motivation

Alpine landscapes are constantly changing, not only in time but also over space. The understanding of spatiotemporal processes and their interrelations is central to the understanding of the complex behavior of real world systems (Pang and Shi, 2002). Relevant processes might span over several temporal and spatial scales. Therefore, tools for modeling, analyzing, and visualizing such processes should also be able to operate on diverse spatial and temporal scales. What kind of tools should be considered to meet these requirements? *Geographic Information Systems* (GIS) provide powerful functionality for spatial analysis, data integration and storage (Nyerges, 1993) and *Virtual Reality* (VR) systems offer interactive virtual fly-through facilities with highly photo-realistic content (Duchaineau et al., 1997; Meyer et al., 2001). These spatially oriented systems lack the ability to represent temporal dynamics and their concepts of landscape are static (Peuquet and Niu, 1995). GIS are very large systems tending to be monolithic, and therefore costly to combine with other systems (Preston et al., 2003). On the other hand, *temporal simulation systems* (TSS) support the simulation of static and in particular dynamic dependencies. Due to the hierarchical structure of state-of-the-art simulation

* Corresponding author. Tel.: +41 1 635 52 57; fax: +41 1 635 68 48.
E-mail address: disen@geo.unizh.ch (D. Isenegger).

0924-2716/$ - see front matter © 2005 International Society for Photogrammetry and Remote Sensing, Inc. (ISPRS). Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.isprsjprs.2005.10.003

models, the composition of complex systems through the coupling of models is possible. A drawback, however, is that in general the spatial dimension is neglected or only poorly represented (Fedra, 1993).

1.1. 'IPODLAS'—coupling TSS, GIS, and VR

The goal of this project is to combine the paradigms and concepts of the three domains TSS, GIS, and VR and to exploit their particular strengths to improve the representation of spatiotemporal and cross-scale processes taking place in the landscape. To this end we are developing the system IPODLAS: interactive, process oriented, dynamic landscape analysis and simulation. The IPODLAS project develops concepts for the information exchange between the different types of subsystems (the TSS, the GIS, and the VR subsystem). Thus an integration of the functionalities of the different subsystems can be achieved. A primary goal is to derive concepts and interfaces for a system that is able to model, analyze, and visualize spatiotemporal and cross-scale processes. The focus is to determine the characteristics and functionalities a system like IPODLAS must include. Three case studies provide realistic data and models supporting the development of IPODLAS. To cover the broad range of possible requirements within landscape analysis, representative case studies have been chosen from very different realms, such as insect population dynamics, wildland fire modeling, and human infrastructure modeling.

1.2. Objectives

This paper focuses on the workflow from a user's requirement through to concepts specifying the architecture of a software system. The workflow comprises of a collection of functional requirements of users within use cases (cf. Section 3.2.1), functionality listings, the design of a software architecture supporting these requirements and implementation of a prototype. Aside from software architecture design, we also aim to present the derivation of requirements and concepts that are important for the design of a system with the requirements of IPODLAS in an exemplary manner.

2. Issues of combination of GIS, VR, and TSS

Taking into account the, in some aspects, complementary strengths and weaknesses of the respective subsystems it seems to be a promising approach to combine them into a common framework. By combining concepts and paradigms of the three domains and exploiting their particular strengths, the investigating of spatiotemporal and cross-scale landscape processes forming the landscape can be improved. The benefits of combination and potential synergies of GIS and TSS (Bernhard and Krueger, 2000; Brimicombe, 2003; Fedra, 1993, 1996; Goodchild et al., 1996; Raper and Livingstone, 1995; Vckovski, 1998) as well as of GIS and VR (Camara et al., 1998; Huang et al., 2001; Lindstrom et al., 1997; Pajarola and Widmayer, 2001) are widely acknowledged. Combining the 'trio' GIS, VR, and TSS promises gains through cross-fertilization and mutual support, but it is conceptually and technologically complex. One underlying core problem is the differing data models used in GIS and TSS (Aspinall and Pearson, 2000; Bennett, 1997; Fedra, 1993, 1996). In the GIS, the data model is centered on representations of the geographical space, the objects located there and their relationships to each other. The focus is on location and topology. TSS data models are designed to model processes, their states, and throughputs of quantities. GIS is designed to model static representations, whereas TSS specializes in model dynamics (Fedra, 1996). These differing emphases result almost necessarily in different conceptual and technological structures (Brimicombe, 2003).

2.1. GIS functionality used

For TSS, the principal benefit of being linked to a GIS is gaining the ability to deal with large volumes of spatially oriented data. Major environmental tasks such as inventory, assessment, management, and prediction in diverse research areas such as atmospheric modeling, land surface–subsurface modeling, and ecological systems modeling can be supported with GIS functionality. The primary modes of GIS usage in practical applications are map, query, and model. The map mode offers to browse information using standard methods such as pan and zoom. In the query mode, the user queries information about locations and/or phenomena. Finally, in the model mode, GIS supports model usage. Although the usage of all primary GIS functionality data entry/capture, data storage/management, data manipulation/analysis, and data display/output can be beneficial to a TSS (Nyerges, 1993), GIS is frequently used only as a preprocessor to prepare spatially distributed input data and as a post-processor to display and analyze model results (Bennett, 1997; Brimicombe, 2003; Fedra, 1993, 1996).

2.2. Integration strategy

The degree to which different systems should be coupled has been and still remains a subject of

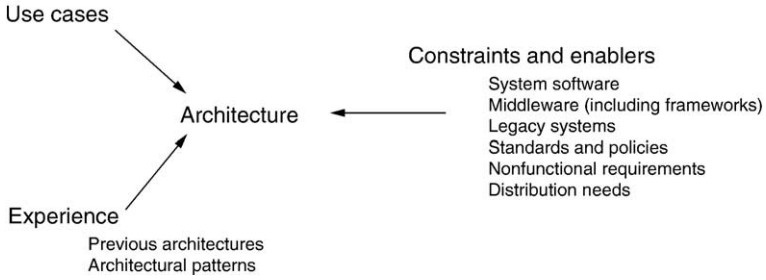

Fig. 1. Requirements and constraints influencing the architecture (Jacobson et al., 1999).

investigation (Brandmeyer and Karimi, 2000; Brimicombe, 2003; Fedra, 1993, 1996; Nyerges, 1993). Brimicombe (2003) suggests what he calls a 'maturing typology' of GIS and environmental model integration ranging from one-way data transfer and loose coupling over shared and joined coupling to tool coupling. The appropriate integration strategy depends on the properties and aims of the respective projects. There are always tradeoffs between contradictory goals, e.g., between efficiency and the flexibility of a system or between the ease of use and the costs of development (Fedra, 1996).

2.3. Interoperability initiatives

In the GIS domain, the OpenGIS Consortium (OGC) (OGC, 2005) defines platform independent, generic interfaces making up a framework supporting interoperability for GIS components (Bernhard and Krueger, 2000; Buehler and McKee, 1998), but it fails to define explicitly the representation of temporal aspects (Schulze et al., 2002). In contrast in the TSS domain, the High Level Architecture (HLA) (IEEE, 2000) provides a framework for distributed time-variant simulation processes. HLA is a federation approach and focuses on interoperability and reuse of simulation. HLA however lacks the support of spatial applications, hence its shortcomings can be considered complementary to the ones of the OGC standards. Simulation models based on the theory of modeling and simulation, discrete event systems specification, and knowledge-based simulation methodologies (De Vasconcelos et al., 2002; Zeigler, 1976, 1990) support due to their hierarchical structure the composition of complex systems through the nesting of models. Although both interoperability approaches remain limited to their respective domains, OGC and HLA provide a promising foundation upon which the integration of the domains GIS and TSS might be built (Schulze et al., 2002).

The Institute of Electrical and Electronics Engineers (IEEE) defines interoperability as the ability of two or more components to exchange information and to use the exchanged information (IEEE, 1990). Emerging concepts from the IT domain provide possibilities to deal with the technical issues associated with integration of different systems. These are, for example, layered architectures used in distributed computing (Ghezzi et al., 2003), the idea of web service trading (Reference Model for Open Distributed Processing, ISO/IEC 10740) (Schulze et al., 2002), or XML-based languages. The application of the client–server paradigm to a Web Map Service or Web Feature Service is an example of 2-tiered architecture (Ghezzi et al., 2003). The addition of a mediating layer is fundamental for 3-tiered architectures such as CORBA[1] (OMG, 1999). The coupling of different GIS or DBMS (Bergmann et al., 2000a,b; Preston et al., 2003) are integration examples from within the GIS domain, the mediation-based framework of Yates and Bishop (1997) and the layered architecture of Bernhard and Krueger (2000) integrates modeling systems and GIS. XML-based languages are used to capture not only formats but also to describe semantics of the information exchanged (Bergmann et al., 2000b; Preston et al., 2003).

3. Methods and materials

3.1. Unified software development process

The ***U**nified Software Development **P**rocess* (UP) (Jacobson et al., 1999) has been used to develop IPODLAS. The goal of the UP is to transform user requirements into a software system. As Fig. 1 shows, *Use cases* (cf. Section 3.2.1) are applied to determine the

[1] Common Object Request Broker Architecture (http://www.corba.org/).

functional requirements. *Constraints and enablers* summarize conditions which must be taken into account when designing a software system (Jacobson et al., 1999).

3.2. Constraints and enablers

3.2.1. Case studies and use cases

Case studies provide real-world data from the test area(s) and simulation models supporting the development of IPODLAS. They help to reduce complexity and act as a test bed for the concepts and applications being developed. Case studies also help in communicating results to potential end-users.[2] A use case specifies a concrete scenario from a case study. All use cases together jointly make up the use case model, which may cover the complete functionality of the planned system. A user definition specifies the general intentions of the user which influences his or her requirements for a software system (Jacobson et al., 1999).

3.2.2. Legacy systems

The applications listed below were chosen because they provide functionality typical of applications in their particular domains and required by IPODLAS to satisfy the requirements captured in the use cases and defined in the functionality lists. RAMSES (Research Aids for Modelling and Simulation of Environmental Systems)[3] has been evaluated to be one of the most appropriate TSS for the needs of IPODLAS (Giorgetta, 2002). RAMSES supports modeling and interactive solving of non-linear differential equations, difference equations, and discrete event systems in any combination (Fischlin, 1991). GRASS GIS (Geographic Resources Analysis Support System)[4] is an open source GIS with raster, topological vector, image processing, and graphics production functionality that operates on various platforms (Neteler and Mitasova, 2002). The subsystem chosen to represent the virtual reality domain is VTP (Virtual Terrain Project).[5] Its goal is to facilitate the creation of tools for interactive, 3D visualization of the earth by bringing together the domains of GIS, visual simulation, surveying and remote sensing.

3.2.3. Standards, policies, and languages

To support the interchangeability of the subsystems, we divide the whole system into subsystems, that is, the design is modular. The modules are determined by high cohesion within the module and low coupling between the modules (Ghezzi et al., 2003). They expose their interfaces and hide their implementation (Preston et al., 2003). The resulting reduction of the communication load between subsystems limits the dependencies between the individual subsystems and therefore supports their interchangeability. Applying interoperability standards to the design of the interfaces using standard communication protocols facilitates compliance of other standard components and therefore the interchangeability of components, augmenting the stability of the interfaces. An example for this is the use of a language of the XML family (XML, 2004) and the application of internet sockets (Stevens, 1990).

The Geography Markup Language GML (Lake et al., 2004; OGC, 2003) is an XML (XML, 2004) extension for encoding the modeling, transport, and storage of the spatial and nonspatial properties of geographic features. The key concepts used by GML to model the world are drawn from the OpenGIS Abstract Specification(OGC, 1999) and the ISO 19100 series (ISO/TC, 2004). The use of GML is expected to lead to greater interoperability between applications within the GIS world and to facilitate data sharing (Preston et al., 2003). With the advent of GML 3.0 (OGC, 2003) the use of temporal information and dynamic features is supported, i.e. there are structures to store and transport temporal information (Preston et al., 2003). A major disadvantage of XML-type languages is the inflated data volume due to additional metadata and the use of a text-based format for encoding binary data. In addition, the parsing of the XML data makes the computer performance critical. Compressing the data and transferring binary data separated from the format description can relief this problem to a certain extent (Hoheisel, 2002).

4. Identifying the required functionality—the IPODLAS approach

This paper addresses methodological aspects. The methodology required to develop the concepts and the IPODLAS prototype are seen as results and therefore presented in Section 4. The concrete setup of the case studies and use case applied in the IPODLAS project is described in detail in the following methodological Sections 4.1–4.3. A concrete example of how to get from the prose description of a use case to the

[2] Knowledge based dynamic landscape analysis and simulation for alpine environments. Full Proposal for SNSF Project Nr. 4048-064432 (http://bscw.geo.unizh.ch/pub/bscw.cgi/d77727/nfp48_scient_final.pdf).

[3] http://www.syseco1.ethz.ch/SimSoftware/SimSoftware.html#RAMSES.

[4] http://www.geog.uni-hannover.de/grass/.

[5] http://www.vterrain.org/.

Table 1
Models at three different scales from three case studies

Case study	Simulation model		
	Small scale	Medium scale	Large scale
LBM (larch bud moth)	M8: Local LBM dynamics (no spatial dimension) (Fischlin, 1982)	M9: combining M8 with migration within valley (Fischlin, 1982)	M10: combining M8 with migration between several valleys (Fischlin, 1983; Giorgetta, 2002)
WLF (wildland fire)	*Local Rothermel: describing fire spread in finite elements* (Rothermel, 1972)	SPARKS: combines the Rothermel model with fire spread models covering surface fire (Schöning, 2000)	FARSITE: combines the Rothermel model with fire spread models covering surface and crown fire and fire spotting (Finney, 2004)
MMI (man-made infrastructure)	*Visualizing single buildings in a street in a village*	*Visualizing village and surroundings*	*Visualizing the whole study area*

Models in italics are not yet implemented on the respective subsystem (Allgöwer et al., 2003).

identification of the required functionality of a system is the subject of the Section 4.4. The resulting software architecture which supports the requirements specified in the use cases is characterized in Section 5.

4.1. Case studies

Three case studies from different domains have been chosen to capture a broad range of requirements. The Larch Bud Moth (LBM) case study represents insect population dynamics, the wildland fire modeling (WLF) case study is an example of an abiotic process, and the modeling of man-made infrastructure (MMI) represents a case study where the human impact on landscape is visualized. These case studies were chosen to include both spatial and temporal aspects and to offer models and associated data across several scales (cf. Table 1) (Price et al., 2005).

4.2. Use cases

The use case model consists of a definition of the users and the description of all use cases. In the IPODLAS project, two types of users are defined to cover diverse requirements, the *pilot*-type and the *expert*-type user. The behavior of the pilot user is characterized by

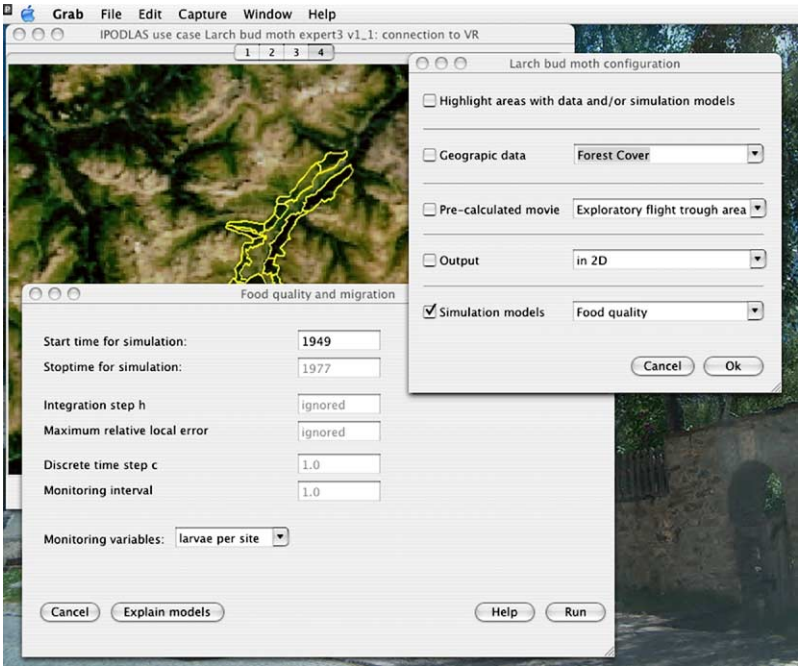

Fig. 2. A screenshot of the IPODLAS GUI defining graphical elements used in an LBM use case.

exploration; she/he flies through the virtual scenery and usually does not change any of the parameter settings but instead uses default configurations when running simulations. In contrast, the expert user is interested in the scientific capabilities of the system; she/he wants to change parameters of the particular subsystems and can plug in new models. Each use case is first described in prose, which defines the particular intentions and the interactions of the user with the system in order to reach the goal of the user described in the related use case. The prose text then is refined in a *sequenced action list*, where the interaction of the user with the system is defined step by step by specifying the input of the user and the response of the system. The graphical definition of the system interface (cf. Fig. 2) helps to specify the state the system is in and the functionality offered.

Among the set of use cases specified, the prospective users select the subset which entails the most important ones. These key use cases may amount only to 5% to 10% of all use cases, but they are the significant ones, as they constitute core system functionality (Jacobson et al., 1999).

4.3. Listing and classifying the required functionality

The use case model consisting of all use cases defines the range of the required functionality that IPODLAS should entail in order to deliver all services the users specify in the various use cases. The functionality recorded in the sequenced action lists forms the basis of the functionality lists describing which functions have to be offered by which subsystem. The functionality list is not only specified from the user's perspective, i.e. seeing the system from outside as a monolith, but conceptually looks to the subsystem level assigning the required functionality to be offered by a particular subsystem. In Table 2, functionality is classified according to estimated implementation efforts of integrating this particular functionality into IPODLAS. The classification of functionality together with the identification of the set of key use cases helps to discover the use cases with the greatest risks of failure.

Table 2
Classification of functionality

Class	Classification of functionality
1.	The required functionality is already implemented in one of the subsystems of IPODLAS.
2.	The required functionality is implemented in another software system.
3.	A solution to offer the required functionality exists in the literature.
4.	An algorithm does not exist in the literature.

4.4. Use case LBM expert 2 (LE2)

Several use cases have been developed in the IPODLAS project, at least one for each user type and for each case study. In the following, a representative example of a use case is described (Price et al., 2005) starting with the definition of the user (cf. Box 1).

Box 1
Use case Bronwyn—the user

> Bronwyn is an 'expert user' of the system IPODLAS. She is a Ph.D. student within the IPODLAS project and wants to use subsystems of the system (GIS, TSS and VR) and the overall system to help her solve research questions regarding the LBM and, then in turn, assist in the development of IPODLAS through provision of data and models and a test bed case study.

The larch bud moth (LBM) population dynamics are spatiotemporal and multi-scaled processes in the Alps. LBM is a forest defoliator causing spectacular damage to larch forests across the Alpine arc, approximately every 9 years (Baltensweiler and Fischlin, 1988). Spatio-temporal dynamics can be modeled by coupling local dynamics models with models of migration between subpopulations at the valley and/or the Alpine Arc scale (Fischlin, 1982, 1983). The prose form of the use case (cf. Box 2) describes the goals of the expert user Bronwyn (Price et al., 2005).

Box 2
Use case LE2—prose description

> Bronwyn is interested particularly in migration of the LBM across the Engadine valley. She wants to see how far LBM migrate per season taking into account wind speed and direction and elements of the landscape which may effect LBM flight such as slope, aspect and local temperature.

Next, the sequenced action list (cf. Table 3) specifies the sequence of interactions of the user with the system (Price et al., 2005).

Table 3
Use case LE2—sequenced action list

Action	Description of action
1.	Bronwyn starts IPODLAS and selects LBM from the list of topics.
2.	IPODLAS shows her the Alpine Arc with highlighted areas, where LBM data can be provided. Bronwyn selects the Upper Engadine valley.
3.	IPODLAS displays a 2D map of the Upper Engadine valley. An additional menu shows several options (geographic data, 3D, simulate, pre-calculated movie). Bronwyn chooses to simulate and see the output in 2D.
4.	Bronwyn chooses a start and stop time (1990, 2000) and otherwise keeps all defaults, then runs the model.
5.	IPODLAS displays a 2D visualisation of the output showing comparative numbers of LBM migrating (departure and landing points).

This use case requires functionality provided from several subsystems. In Table 4, only the actions where the GIS subsystem is involved are specified and classified (in the column 'Class') according to Section 4.3. For the sake of clarity, the actions from the sequenced action list are described in substeps.

The GIS subsystem is only involved in the actions 2, 4, and 5 of the sequenced action list. This can be explained through the division of labor on the subsystem level: storage(s) is accessible by every subsystem and the navigation of the user is handled by the VR subsystem. The functionality listing in Table 4 is typical for the functionality listings of use cases developed for IPODLAS. A major part of the required GIS functionality in the use cases, at least on this level, is standard spatial analysis functionality such as map algebra and map overlay or data integration such as joining attribute data (in textual form) and spatial data. When implementing these use cases the challenges that are occurring at this stage of IPODLAS development is not (yet) missing GIS functionality, but rather the communication between the subsystems. The action numbers 4a, 4d, 5a, and 5c can be classified in functionality classes 2, 3, or 4 depending on the conceptual and technical complexity of the chosen solution to provide this functionality. As an example, for step 4d classification of this task into class 2 could mean that data is sent only as simple text file to the requesting subsystem, while class 3 indicates a more advanced solution such as the automatic encoding of spatiotemporal data in GML3.x for transmission (this will be explained in Section 5.2.). On a conceptual level, in action numbers 4a and 5a, the control flow is affected, i.e. seamless access of functionality is required. Action numbers 4d and 5c address the data flow, i.e. seamless data access. In a study analyzing interoperable and distributed GIS, Bergmann et al. (2000b) present similar findings. The ability of interaction of components through information exchange, in particular seamless data access and access to remote methods is a major requirement to move towards interoperable GIS.

Table 4
Use case LE2—required GIS functionality

Action	Required functionality	Class	Applied standard GIS functionality
2a.	Receiving request to provide areas with available data	2/3/4	Communication/information exchange between subsystems
2b.	Selecting data from storage(s): areas with available data	2	Connection to storage(s) and retrieving data from storage(s)
2c.	Notifying the requesting subsystem about available data via kernel	2/3/4	Communication/information exchange between subsystems
4a.	Receiving request concerning forest and in particular larch distribution, calculating slope and aspect, and wind simulation	2/3/4	Communication/information exchange between subsystems
4b.	Selecting data from storage(s): forest data, larch data, temperature data, DTM	2	Connection to storage(s) and retrieving data from storage(s)
4c.	Calculating: larch per hectare, forest area per hectare, temperature distribution	1	Map algebra, Map overlay, clipping
	Calculating slope, aspect	1	Slope, aspect calculation from DTM
	Simulating Wind speed and direction	2	Querying wind model
4d.	Notifying the requesting subsystem about available data via kernel	2/3/4	Communication/information exchange between subsystems
5a.	Receiving request to transform tabular simulation output to raster	2/3/4	Communication/information exchange between subsystems
5b.	Transform tabular simulation output in raster	1	Join attribute data with spatial data
5c.	Notifying the requesting subsystem about available data via kernel	2/3/4	Communication/information exchange between subsystems

5. Software architecture

The overall goal is to develop an architecture that makes the system resilient to change or change tolerant. The software architecture includes the most important static and dynamic aspects of the design of a system. It focuses on significant structural elements as well as on the interactions that occur among these elements via interfaces (Jacobson et al., 1999). Due to the iterative nature of software development, in some chapters of this paper the individual steps of architecture refinement are split into different phases of development (e.g., early phase, advanced phase). Owing to the modular design of IPODLAS, the dependencies between different architectural aspects are limited and therefore subsystems in different phases should interact smoothly and seamlessly with another. This means that in some architectural aspects the features planned in the advanced phase can be implemented while other aspects remain in the early phase.

5.1. Development of the software architecture

To illustrate the iterative approach, this section outlines some prototypes of IPODLAS which demonstrate the evolution of important parts or phases of the software architecture. The first prototype, the "*Intelligent Tree*", presents a simple interaction model of the subsystems, while the second prototype *Cross-implementation* points at the benefits of the interaction of the subsystems. Section 5.1.3. discusses an example for inter-process communication.

5.1.1. The 'intelligent tree'

The earliest version of the IPODLAS prototype was built to model the growth of an '*intelligent tree*' (Fischlin et al., 2002). The tree was considered as intelligent because it is aware of its location and therefore its growth conditions. In this prototype the user can specify the place where a tree is to grow by clicking with the pointing device within the VR subsystem (cf. Fig. 3). The GIS delivers data related to the habitat conditions (elevation, slope, and aspect) at the chosen location calculated from the DEM, while the TSS subsystem calculates the growth rate according to climate conditions, which are determined by elevation. Tree growth is visualized in a stepwise fashion in the VR subsystem. Data flow is handled by file exchange, while control flow is based on semaphores. Each subsystem is only allowed to access its respective input data file when the semaphore associated with this data file exists. Then the active subsystem has exclusive file access. After termination of all file accessing operations of the active subsystem the respective *Ready-semaphore* R_i associated with the output file of the active subsystem is generated to notify the other waiting subsystems. This initial prototype served as test bed for realizing a concrete division of labor among the subsystems and to test specific communication means such as file-coupling (Fischlin et al., 2002). Exchanging information via files is an acceptable when advantages of simplicity outweigh performance losses. This depends on the operating system's characteristics of generating, reading, writing, and destroying files. A drawback is that synchronization of file access by semaphore files is not very flexible in comparison to a process-based approach with a central coordination process, i.e. synchronization of a more complex system by semaphore is quickly prohibitively challenging.

5.1.2. Cross-implementation

In a subproject known as *Cross-implementation* (Isenegger et al., 2004) a fire simulation model already implemented in the GIS (*Sparks* (Schöning, 2000), cf. Table 1) was implemented in the TSS subsystem and an LBM simulation model already implemented in the TSS

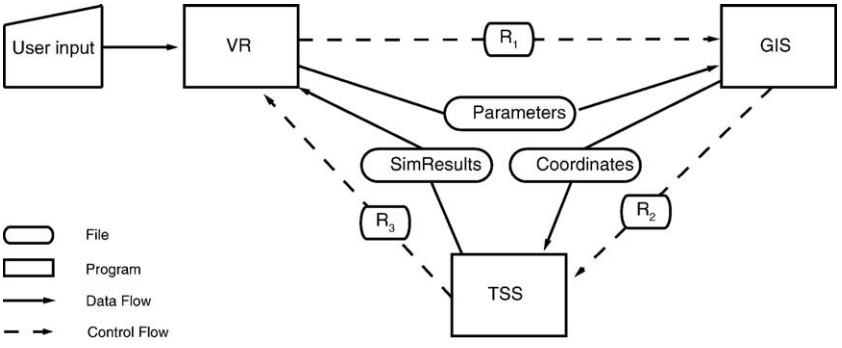

Fig. 3. Overview of the "*Intelligent Tree*" Architecture. R_i *(Ready-semaphore*) represents the respective semaphore files. *Parameters*, *Coordinates*, and *SimResults* are the data exchange files.

subsystem was implemented in GIS (*LBM8* (Fischlin, 1982; Fischlin and Baltensweiler, 1979), cf. Table 1). The aim of this subproject was to discover capabilities and limits of the particular system dealing with problems for which the systems are not designed. That is, the GIS was challenged with a simulation model computing mainly temporal processes and the TSS subsystem with processes with a strong spatial aspect. In this project an ArcInfo 8.1 workstation was used as the GIS subsystem and RAMSES as the TSS subsystem. While RAMSES provides libraries offering sophisticated mathematical and simulating capabilities, it lacks spatial functions, particularly for displaying geo-referenced data, spatial analysis, and storage of large volumes of spatial data. ArcInfo only has limited to no temporal functionality in comparison to RAMSES and besides its performance disadvantages the macro language AML does not support higher programming concepts. Aside from providing the somewhat trivial insight that applications best deal with problems for which they are designed, this project highlighted needed functionality and which subsystem should best provide this functionality (Isenegger et al., 2004).

5.1.3. Socket communication

Fig. 4 illustrates that the TSS subsystem can be controlled over a network applying the client/server approach. A client can trigger the TSS subsystem to start and stop simulation model runs, change the active simulation model and transfer results encoded in XML (Bergamin, 2004). The inter-process communication between the XML-RPC-Server and the TSS subsystem uses AppleEvents, while the communication between the client and the server is done with sockets using XML-RPC protocol (UserLand Software, 2003).

5.2. Current software architecture

As the use cases showed, the interaction of the subsystems follows a certain sequence of (inter)actions requiring some form of synchronization. As a central coordination subsystem, the IPODLAS kernel (cf. Fig. 5) provides this functionality: all control flow is managed by the kernel. The IPODLAS managed storage holds metadata describing the data available to IPODLAS. It is closely linked to the second persistent storage of the IPODLAS system, the GIS storage. All data flow from the storages to the subsystems is managed by direct interaction to avoid the coordination subsystem becoming a bottleneck.

Modularity of the architecture is enhanced by the coordination subsystem, that is, the IPODLAS kernel, limiting the number of interfaces needed for communication between the subsystems. The role of the kernel as the only communication interface between the subsystems means that changes in the interaction of the subsystems or even the exchange of a subsystem, for example the use of another GIS, need only to be registered in the IPODLAS kernel. An exception to the strict modularity and the separation of control and data flow, is the interface between VR and the GUI due to possible heavy communication load between these two subsystems and to real-time requirements.

As reported by Leclercq et al. (1996), each subsystem must map the parts of its data model that are essential for IPODLAS onto the canonical data model of the IPODLAS kernel. To be able to access the

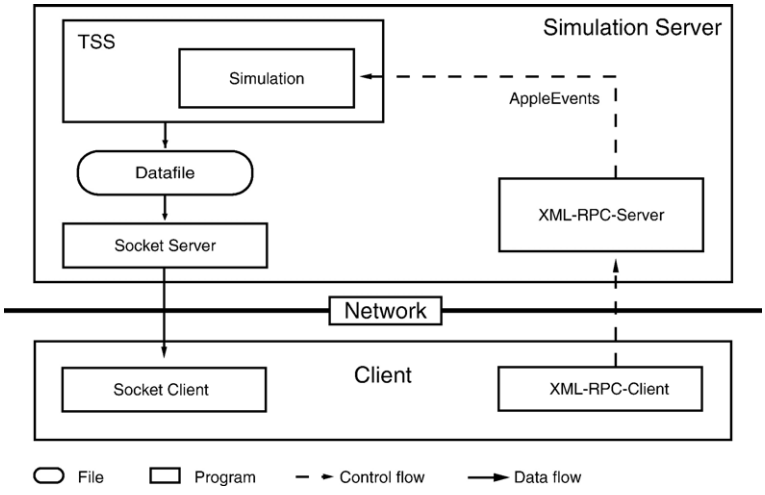

Fig. 4. The socket client/server model of the TSS subsystem (after Bergamin, 2004).

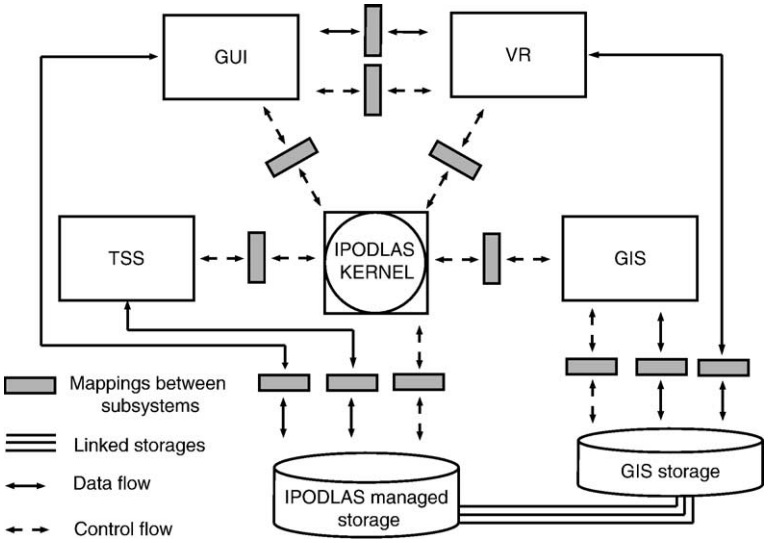

Fig. 5. A schematic view of the IPODLAS software architecture.

functionality of the subsystems functional mapping is also mandatory. In an initial development phase the central coordination process can be kept rather simple, restricted to sequencing the interactions of the subsystems. An advanced solution provides a mediating kernel such as the ones presented in Zaslavsky et al. (2000) and Savary and Zeitouni (2003) receiving requests, dispatching them to the appropriate system, and providing feedback to the user.

5.2.1. Data exchange

In Fig. 6 an example of communication between the GIS subsystem and the TSS subsystem is outlined. The TSS subsystem needs to know the terrain aspect of a certain area for a simulation task and thus sends a request to the kernel (step 1 in Fig. 6), written to a socket connection. The kernel listens to the socket connection, gets the request and dispatches it to the appropriate subsystem, the GIS (step 2). The GIS reads the digital elevation model (DEM) from the storage (step 3) and calculates the terrain aspect values for the DEM (step 4), serializes the result and writes it back to the storage (step 5). Then the requesting subsystem, the TSS, receives the notification through the kernel (step 6 and step 7) of where the aspect data is and reads this information (step 8).

5.2.2. GML 3—temporal aspects

GML 3 specifies the schemas *temporal.xsd* and *dynamicFeatures.xsd* to represent temporal issues. The former schema defines primitives and properties for representing temporal instants and periods. The latter schema allows definition of elements and types to model dynamic features. A *DynamicFeature*, aside from time-invariant properties, entails a *history* property to express the historical development of the feature. The *history* associates the feature with a sequence of time slices which include the dynamic properties of the feature (Lake et al., 2004). Employing those features of GML3 to the LBM case study (cf. Section 4.) augments

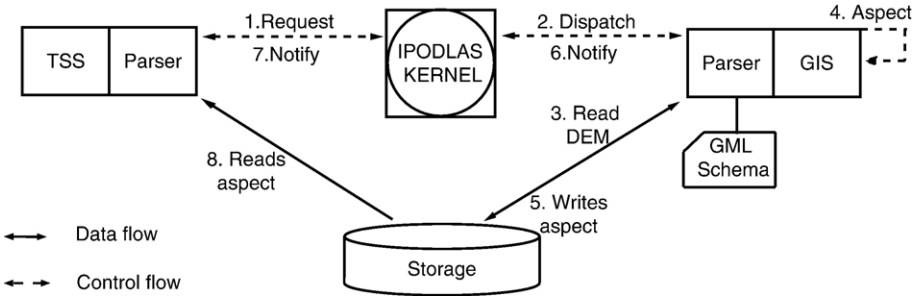

Fig. 6. Communication between the TSS and the GIS for the example of terrain aspect computation.

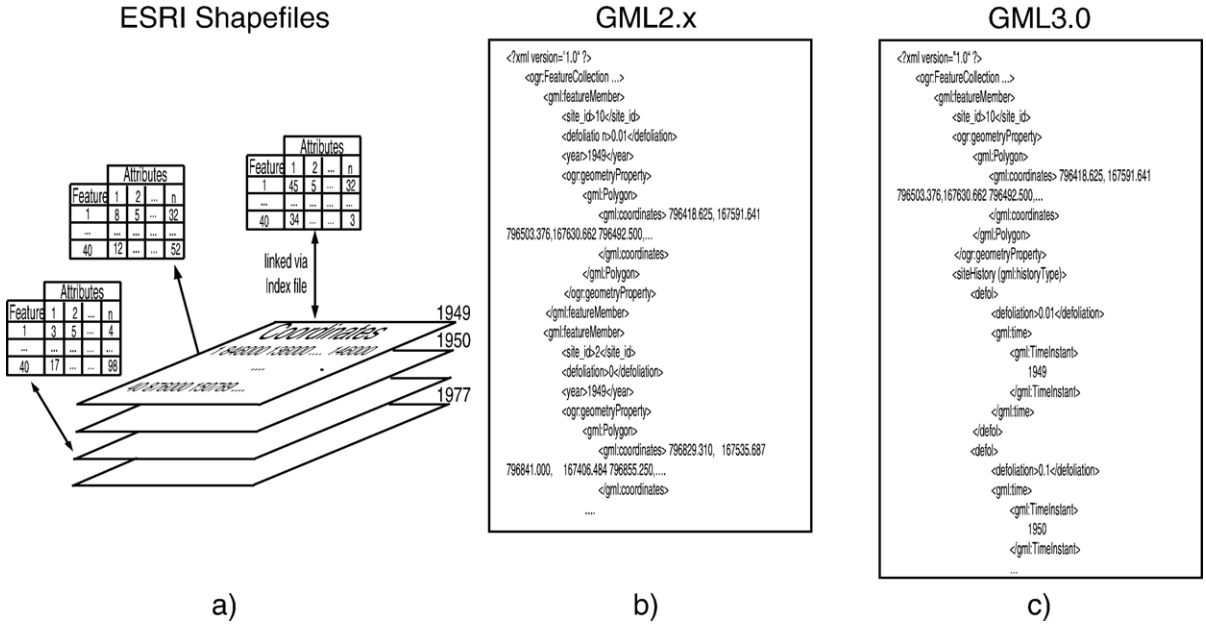

Fig. 7. a) ESRI shapefiles showing areas with LBM infestation of the years 1949 to 1977. b) The same data encoded in GML2.x. with repetitive encoding of time-invariant properties. c) In GML3.0, opposed to GML2.x, time-invariant properties are encoded only once while the time-variant properties are represented in *history* elements containing different values for each year.

the expressiveness of the data structure. The standard GIS representation of the LBM topic is that for each year a dataset exists in a snapshot-like fashion containing the information about the study area (cf. Fig. 7a). The temporal elements of GML 3 enrich the data structure to entail dynamic subsets of properties. A research area of the LBM case study is represented by a *DynamicFeature.* Time-invariant properties of a study area are for example its location, perimeter, and the coordinate system. The time slices comprise time-varying properties such as defoliation values, the amount of LBM larvae, and the year (cf. Fig. 7c). The use of the temporal features of GML3 leads to a more economic representation (Lake et al., 2004) due to the concentration of often voluminous geographic data in only one place. On the other hand, this representation supports a more object-based view of the research area, which is mapped here as one object with time-invariant properties and series of time-varying properties.

6. Discussion

6.1. The IPODLAS approach

To challenge the capabilities of IPODLAS to be able to deal with dynamic and cross-scale processes, case studies from different domains with diverse user types provide data from dynamic processes and different simulation models which act in space and time. Since simulation models acting on different scales can be coupled, the representation of cross-scale process is supported. In this project where integration of legacy systems is an important part of the development, the functionality listings are a concise and structured instrument to determine and evaluate integration efforts of existing and lacking functionality required by the users. Use of the structured approach of the UP (Jacobson et al., 1999) is a formal and transparent way for both users and developers to support the determination and description of user requirements and to move from requirements to a software system.

6.2. Software architecture

Due to the modular architecture of the system a stepwise refinement and enhancement of the system can be achieved, which allows for separate development of different aspects and therefore a smooth interaction of subsystems that are in different phases of their development. Another benefit of a modular design is the enhanced reusability (Preston et al., 2003), extensibility, and scalability of the system (Bergmann et al., 2000a; Wang, 2000). Similar to Bergmann et al. (2000b) and Bernhard and Krueger (2000) the layered architecture limits the interdependencies between the subsystems.

Additionally, as in Bergmann et al. (2000b), a central coordination process synchronizes the exchange of information between the subsystems.

6.3. Coupling TSS, GIS, and VR

Current approaches to coupling TSS and GIS and coupling GIS and VR are that GIS provide a platform for data integration, model parameter determination and cartographic visualization. TSS provides temporal capabilities and allows GIS to go beyond inventory and thematic mapping (Sui and Maggio, 1999). VR offers realistic representation and interactive exploration (Camara et al., 1998). By combining all *three* systems TSS, GIS, and VR IPODLAS moves a step further towards tool coupling following the hierarchical typology of integration of Brandmeyer and Karimi (2000), which is a networked modeling framework having integral subsystems wrapped within a common user interface. Within the framework subsystems share data and storage, and the common user interface supports seamless access to the functionality of all subsystems (Brimicombe, 2003).

6.4. GML

The encoding of the information exchange between the GIS subsystem and the central coordination process through GML is independent of the platform, operating system, language, and the data transfer protocol. Parsers can validate data structure as defined by the XML Schema. GML is an open structure providing the possibility of further enhancement. However, an increased data volume due to the tag structure has to be accepted. Compared to previous versions, GML 3.0 provides extensions covering events, histories, and timestamps (Lake, 2001; Lake et al., 2004). This offers the required data structures to prevent loss of semantics and enrich and facilitate the information exchange between the GIS and the TSS subsystem.

6.5. Lessons learned

The prototype simulating the '*intelligent tree*' showed that the file-based information exchange synchronized with semaphore files is straightforward, but becomes quickly complicated when the synchronization is complex (cf. Section 5.1.1.). The main outcome of the Cross-Implementation approach was that both subsystems (TSS and GIS) deal well with the problems they are designed for while problems occur when conducting research not explicitly supported by the systems (cf. Section 5.1.2.). This confirms the hypothesis that when doing joint research each subsystem can bring in its strengths and avoid its weaknesses. Therefore, IPODLAS can benefit from the complementary capabilities of the respective subsystems (Isenegger et al., 2004).

7. Conclusion and outlook

This paper presents the software architecture of the IPODLAS project, which aims to bring two different views and conceptualizations of views of the world – the spatial and the process-oriented – to closer proximity. Developing use cases within three diverse case studies and the derivation of listings of functionality are a systematic means to capture functional specification of requirements. To achieve the highest level of integration according to the classification of Lilburne (1996) the software architecture must be refined further to fully integrate the user interface, data and functionality of the subsystems TSS, GIS, and VR. The future of IPODLAS development is the component-based paradigm, aiming for interoperable components with exposed interfaces and hidden implementations using component-oriented middleware technologies such as CORBA (OMG, 1999), DCOM (Sessions, 1998) or EJB (SunMicroSystems, 2001). Much of the kernel functionality can be provided by an application server, which receives requests and distributes tasks to the appropriate subsystem(s), while XML-RPC (UserLand Software, 2003) or SOAP (W3C, 2003) can be used to exchange information between the subsystems. The same criteria apply to offering the functionalities of IPODLAS for geospatial services as for standard web services. Thus, IPODLAS must provide both a catalog service with metadata describing the services offered and the interfaces themselves on the syntactic and the semantic level and, furthermore, must support access via HTTP[6] and standards such as WSDL[7], UDDI[8] and SOAP to ensure interoperability (Riedemann and Timm, 2003).

Acknowledgements

This research was partially supported by the Swiss National Science Foundation under contract no. 4048-064432. The IPODLAS project is part of the National Research project NRP 48 "*Landscape and Habitats in*

[6] Hypertext Transfer Protocol.
[7] Web Service Description Language.
[8] Universal Description, Discovery and Integration.

the Alps" (NFP48, 2004) of the Swiss National Science Foundation (SNSF, 2005). The helpful comments by two anonymous reviewers are gratefully acknowledged.

References

Allgöwer, B., Fischlin, A., Frei, U., 2003. Use case approach road map, Internal report. http://bscw.geo.unizh.ch/bscw/bscw.cgi/d78284/use%20case%20approach%20roadmap.pdf (accessed October 10, 2005).

Aspinall, R., Pearson, D., 2000. Integrated geographical assessment of environmental condition in water catchments: linking landscape ecology, environmental modelling and GIS. Journal of Environmental Management 59 (4), 299–319.

Baltensweiler, W., Fischlin, A., 1988. The Larch Budmoth in the Alps. Dynamics of Forest Insect Populations. Plenum, New York.

Bennett, D., 1997. A framework for the integration of geographical information systems and modelbase management. International Journal of Geographical Information Science 11 (4), 337–357.

Bergamin, J., 2004. Schnelle Datenübertragung für verteilte Simulation und Visualisierung. MSc Thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, http://bscw.geo.unizh.ch/pub/bscw.cgi/d77772/datenAT_SimuVisu_DABergamin_04.pdf (accessed October 10, 2005).

Bergmann, A., Breunig, M., Cremers, A., Shumilov, S., 2000a. Towards an interoperable open GIS. In: Norrie, M., Laurini, R., Spaccapietra, S. (Eds.), Proceedings International Workshop on Emerging Technologies for Geographical Information Systems for Geo-based Applications, Ascona, Switzerland, May 22–25, pp. 283–296.

Bergmann, A., Breunig, M., Cremers, A., Shumilov, S., 2000b. A component based, extensible software platform supporting interoperability of GIS applications. Proceedings of the Umweltinformatik 2000 — Computer Science for Environmental Protection. Metropolis-Verlag.

Bernhard, L., Krueger, T., 2000. Integration of GIS and spatio-temporal simulation models: interoperable components for different simulation strategies. Transactions in GIS 4 (3), 197–215.

Brandmeyer, J.E., Karimi, H.A., 2000. Coupling methodologies for environmental models. Environmental Modelling and Software 15 (5), 479–488.

Brimicombe, A., 2003. GIS, Environmental Modelling and Engineering. Taylor & Francis, London. 312 pp.

Buehler, K., McKee, L., 1998. The OpenGIS Guide. Open GIS Consortium Technical Committee. Wayland.

Camara, A.S., Neves, J.N., Muchaxo, J., Fernandes, J.P., Sousa, I., Nobre, E., Costa, M., Mil-Homens, J., Rodriques, A.C., 1998. Virtual environments and water quality management. Journal of Infrastructure Systems 4 (1), 28–36.

De Vasconcelos, M.J.P., Goncalves, A., Catry, F.X., Paul, J.U., Barros, F., 2002. A working prototype of a dynamic geographical information system. International Journal of Geographical Information Science 16 (1), 69–91.

Duchaineau, M., Wolinski, M., Sigeti, D., Miller, M., Aldrich, C., Mineev-Weinstein, M., 1997. ROAMing terrain: real-time optimally adapting meshes. IEEE Visualization, vol. 97. Phoenix, USA, pp. 81–88.

Fedra, K., 1993. Gis and Environmental Modeling. Environmental Modeling with GIS. Oxford University Press, New York, pp. 35–50.

Fedra, K., 1996. Distributed Models and Embedded GIS: Integration Strategies and Case Studies. GIS and Environmental Modeling: Progress and Research Issues. GIS World Books, Fort Collins, CO, pp. 413–418.

Finney, M., 2004. FARSITE, fire area simulator—model development and evaluation. Research paper RMRS; RP-4. U.S. Dept. of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, UT (324 25th St., Ogden 84401), 47 pp.

Fischlin, A., 1982. Analyse eines Wald–Insekten-Systems: Der subalpine Lärchen-Arvenwald und der graue Lärchenwickler Zeiraphera diniana GN. (Lep., Tortricidae). Diss. ETH Thesis, Swiss Federal Institute of Technology, Zürich, Switzerland. 294 pp.

Fischlin, A., 1983. Modelling of Alpine valleys, defoliated forests, and larch bud moth cycles: the rôle of migration. In: R.H. Lamberson (Ed.), Mathematical models of renewable resources. Humboldt State University, Mathematical Modelling Group, University of Victoria, Victoria, B.C., Canada, pp. 102-104.

Fischlin, A., 1991. Interactive modeling and simulation of environmental systems on workstations. In: Möller, D.P.F., Richter, O. (Eds.), Analysis of Dynamic Systems in Medicine, Biology, and Ecology. Informatik-Fachberichte. Springer, Berlin, pp. 131–145. a.o.

Fischlin, A., Baltensweiler, W., 1979. Systems analysis of the larch bud moth system: Part I. The larch–larch bud moth relationship. Mitteilungen der Schweizerischen Entomologischen Gesellschaft 52, 273–289.

Fischlin, A., Price, B., D. Isenegger, Porchet, P., Allgöwer, B., Frei, U., 2002. Use Case TreeGrowth for IPODLAS Prototype 0.0, Internal report. Workshop Nov 2002, internal report, http://bscw.geo.unizh.ch/pub/bscw.cgi/d77716/IPODLAS_UC_TreeGrowth_v0.6.pdf (accessed in October 10, 2005).

Ghezzi, C., Jazayeri, M., Mandrioli, D., 2003. Fundamentals of Software Engineering. Prentice Hall, Upper Saddle River, N.J. xx, 604 pp.

Giorgetta, F., 2002. Integration von Raum und Zeit in ein Landschaftsanalyse und Simulationssystem: Systemtheoretische Grundlagen und Evaluation gängiger Systemsimulationsprogramme. Semesterarbeit Thesis, Systems Ecology, Institute of Terrestrial Ecology, Swiss Federal Institute of Technology, Zurich, Switzerland, 54 p, http://bscw.geo.unizh.ch/bscw/bscw.cgi/d77722/fgiorgetta_2002.pdf (accessed October 10, 2005).

Goodchild, M., Steyaert, L., Parks, B., 1996. GIS and environmental modeling : progress and research issues. GIS World Books, Fort Collins, CO. xvii, 486 pp.

Hoheisel, A., 2002. SWIM meets XML— The Man Model Measurement (M3) project. http://mmm.first.fhg.de/papers/hoheisel_m3_iemss2002.pdf2002(accessed October 10, 2005).

Huang, B., Jiang, B., Li, H., 2001. An integration of GIS, virtual reality and the Internet for visualization, analysis and exploration of spatial data. International Journal of Geographical Information Science 15 (5), 439–456.

IEEE, 1990. IEEE Standard Computer Dictionary: A compilation of IEEE Standard Computer Glossaries. Institute of Electrical and Electronics Engineers, Inc., New York.

IEEE, 2000. 1516-2000 IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA) — framework and rules. http://standards.ieee.org/catalog/olis/compsim.html (accessed May 1, 2005).

Isenegger, D., Price, B., Wu, Y., 2004. Crossimplementation of wildland fire and larch bud moth models in GIS and Ramses, Internal report. http://bscw.geo.unizh.ch/pub/bscw.cgi/d77674/x_implementation.pdf (accessed October 10, 2005).

ISO/TC, 2004. International Organization for Standardization/Technical committees 211. http://www.isotc211.org/ (accessed October 10, 2005).

Jacobson, I., Booch, G., Rumbaugh, J., 1999. The Unified Software Development Process. Addison-Wesley, Reading, Massachusetts.

Lake, R., 2001. GML 2.0 — Enabling the Geo-spatial Web. http://www.geojava.net/company/galdos/articles/GML3.htm (accessed October 10, 2005).

Lake, R., Burggraf, D., Trinic, M., Rae, L., 2004. GML Geography Mark-Up Language. John Wiley & Sons, Southern Gate, Chichester, West Sussex.

Leclercq, E., Benslimane, D., Yetongnon, K., 1996. A distributed object architecture for interoperable GIS. Ninth International Conference on Parallel and Distributed Computing Systems.

Lilburne, L., 1996. The integration challenge. Proceedings 8th Annual Colloquium of the Spatial Information Research Centre, Dunedin, New Zealand, pp. 85–94.

Lindstrom, P., Koller, D., Ribarsky, W., Op den Bosch, H., Hodges, L., Faust, N., 1997. An integrated Global GIS and Visual Simulation System, GVU Technical Report 97-07. Atlanta., ftp://ftp.gvu.gatech.edu/pub/gvu/tech-reports/1997/97-07.pdf (accessed October 10, 2005).

Meyer, A., Neyret, F., Poulin, P., 2001. Interactive rendering of trees with shading and shadows, Eurographics Workshop on Rendering. London, England. http://artis.imag.fr/Publications/2001/MNP01/MNP01.pdf (accessed October 10, 2005).

Neteler, M., Mitasova, H., 2002. OPEN SOURCE GIS: A GRASS GIS Approach. Kluwer Academic Publishers, Boston.

NFP48, 2004. National Research Project NFP 48, Landschaften und Lebensräume der Alpen, National Research Project NFP 48.

Nyerges, T., 1993. Understanding the Scope of GIS: Its Relationship to Environmental Modeling. Environmental Modeling with GIS. Oxford University Press, New York, pp. 75–107.

OGC, 1999. The OpenGIS Abstract Specification Overview, Version 4. http://www.opengis.org/techno/specs.htm (accessed October 10, 2005).

OGC, 2003. OpenGIS Geography Markup Language (GML) Implementation Specification, version 3.0. http://www.opengeospatial.org/specs/?page=specs (accessed October 10, 2005).

OGC, 2005. OGC — Open Geospatial Consortium. http://www.opengeospatial.org/2005(accessed October 10, 2005).

OMG, 1999. CORBA components and component model, document orbos/99-02-05. http://www.omg.org/ (accessed October 10, 2005).

Pajarola, R., Widmayer, P., 2001. Virtual geoexploration: concepts and design choices. International Journal of Computational Geometry and Applications 11 (1), 1–14.

Pang, M.Y.C., Shi, W.Z., 2002. Development of a process-based model for dynamic interaction in spatio-temporal GIS. Geoinformatica 6 (4), 323–344.

Peuquet, D.J., Niu, D.A., 1995. An Event-Based Spatiotemporal Data Model (Estdm) for Temporal Analysis of Geographical Data. International Journal of Geographical Information Systems 9 (1), 7–24.

Preston, M., Clayton, P., Wells, G., 2003. Dynamic run-time application development using CORBA objects and XML in the field of distributed GIS. International Journal of Geographical Information Science 17 (4), 321–341.

Price, B., Isenegger, D., Wu, Y., 2005. Use Case Larch Budmoth Expert 2, Internal report. http://bscw.geo.unizh.ch/pub/bscw.cgi/d77703/use_case_LE2_v1_2.pdf (accessed October 10, 2005).

Raper, J., Livingstone, D., 1995. Development of a geomorphological spatial model using object-oriented design. International Journal of Geographical Information Science 9 (4), 359–383.

Riedemann, C., Timm, C., 2003. Services for data integration. Data Science Journal 2, 75–83 (Spatial data usability special section).

Rothermel, R., 1972. A mathematical model for predicting fire spread in wildland fuels. Internal report., USDA Forest Service, Intermountain Forest and Range Experiment Station.

Savary, L., Zeitouni, K., 2003. Spatial data warehouse — a prototype. Electronic Government, Proceedings, vol. 2739, pp. 335–340.

Schöning, R., 2000. Modellierung des potentiellen Waldbrandverhaltens mit einem GIS. Msc Thesis Thesis, University of Zürich, Zürich, Switzerland, http://www.geo.unizh.ch/gis/research/geoprocessing/gp-abst29.shtml (accessed October 10, 2005).

Schulze, T., Wytzisk, A., Simonis, I., Raape, U., 2002. Distributed spatio-temporal modeling and simulation. In: Yuecesan, E., Chen, C., Snowdon, J., Charnes, J. (Eds.), Winter Simulation Conference, San Diego, CA, pp. 695–703.

Sessions, R., 1998. COM and DCOM. John Wiley Press, New York.

SNSF, 2005. Swiss National Science Foundation, http://www.snf.ch/default_en.asp (accessed October 10, 2005).

Stevens, W., 1990. UNIX Network Programming. Prentice Hall P T R, Englewood Cliffs, NJ.

Sui, A., Maggio, R., 1999. Integrating GIS with hydrological modeling: practices, problems and prospects. Computers, Environment and Urban Systems 23, 33–51.

SunMicroSystems, 2001. Enterprise JavaBeans technology. http://java.sun.com/products/ejb/ (accessed October 10, 2005).

UserLand Software, 2003. XML-RPC Home Page, http://www.xmlrpc.com/ (accessed October 10, 2005).

Vckovski, A., 1998. Interoperable and distributed processing in GIS. Taylor & Francis, London. xvii, 230 pp.

W3C, 2003. SOAP Version 1.2. http://www.w3.org/TR/soap/ (accessed October 10, 2005).

Wang, F., 2000. A distributed geographic information system on the common object request broker architecture. GeoInformatica 4 (1), 89–115.

XML, 2004. Applying XML and Web Services Standards in Industry. http://www.xml.org (accessed October 10, 2005).

Yates, P., Bishop, I., 1997. A Method for the Integration of Existing GIS and Modeling Systems, GeoComputation. University of Otago, New Zealand, pp. 191–197.

Zaslavsky, I., Marciano, R., Gupta, A., Baru, C., 2000. XML-based Spatial Data Mediation Infrastructure for Global Interoperability, 4th Global Spatial Data Infrastructure Conference, Cape Town, South Africa, March 13–15. http://www.npaci.edu/DICE/Pubs/gsdi4-mar00/gsdi_iz.html (accessed October 10, 2005).

Zeigler, B.P., 1976. Theory of Modeling and Simulation. Wiley, New York. xxii, 435 pp.

Zeigler, B.P., 1990. Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems. Academic Press, Boston. xvii, 395 pp.